

WebSphere Message Broker



# Message Models

*Version 6 Release 0*



WebSphere Message Broker



# Message Models

*Version 6 Release 0*

**Note**

Before using this information and the product it supports, read the information in the Notices appendix.

**First Edition (September 2005)**

This edition applies to IBM® WebSphere® Message Broker Version 6.0 and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 2000, 2005. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

**About this topic collection. . . . . v**

---

## **Part 1. Developing message models 1**

### **Developing message models. . . . . 3**

Message modeling . . . . .	3
Working with a message set project . . . . .	79
Working with a message set . . . . .	80
Working with a message definition file . . . . .	91
Working with message model objects . . . . .	92
Creating a multipart message . . . . .	117
Linking from one message definition file to another	118
Working with a message category file . . . . .	120
Importing . . . . .	124
Generating information from message sets . . . . .	139

---

## **Part 2. Reference . . . . . 143**

**Message model reference information 145**

Message set preferences . . . . .	145
Message set properties . . . . .	147
Message definition file properties. . . . .	174
Message category properties . . . . .	175
Message model object properties . . . . .	177
Deprecated message model object properties . . . . .	549
Additional MRM domain information . . . . .	673
Additional MIME domain information . . . . .	712
Generated model representations . . . . .	716
Import formats . . . . .	722
Message model wizards . . . . .	734

---

## **Part 3. Appendixes . . . . . 773**

**Appendix. Notices . . . . . 775**

Trademarks . . . . . 777

**Index . . . . . 779**



---

## About this topic collection

This PDF has been created from the WebSphere Message Broker Version 6.0 GA (September 2005) information center topics. Always refer to the WebSphere Message Broker online information center to access the most current information. The information center is periodically updated on the document update site and this PDF and others that you can download from that Web site might not contain the most current information.

The topic content included in the PDF does not include the "Related Links" sections provided in the online topics. Links within the topic content itself are included, but are active only if they link to another topic in the same PDF collection. Links to topics outside this topic collection are also shown, but these attempt to link to a PDF that is called after the topic identifier (for example, ac12340\_.pdf) and therefore fail. Use the online information to navigate freely between topics.

**Feedback:** do not provide feedback on this PDF. Refer to the online information to ensure that you have access to the most current information, and use the Feedback link that appears at the end of each topic to report any errors or suggestions for improvement. Using the Feedback link provides precise information about the location of your comment.

The content of these topics is created for viewing online; you might find that the formatting and presentation of some figures, tables, examples, and so on are not optimized for the printed page. Text highlighting might also have a different appearance.





---

## Part 1. Developing message models

<b>Developing message models</b>	<b>3</b>
Message modeling	3
Message modeling concepts	4
Why model messages?	6
Which message domain and format to use?	7
The message model.	10
Physical formats in the MRM domain	39
Ways to create message definitions	64
Generate model representations	75
Working with a message set project	79
Creating a message set project	79
Deleting a message set project	80
Working with a message set.	80
Configuring message set preferences	80
Opening an existing message set	81
Creating a message set	81
Configuring logical properties: Message sets	83
Working with physical properties	84
Configuring documentation properties: Message sets	89
Deleting a message set.	90
Applying a Quick Fix to a task list error.	90
Working with a message definition file	91
Opening an existing message definition file.	91
Creating a message definition file	91
Deleting a message definition file	92
Working with message model objects.	92
Adding message model objects	93
Configuring message model objects	105
Deleting objects.	117
Creating a multipart message	117
Linking from one message definition file to another	118
Include	119
Import.	119
Working with a message category file	120
Creating a message category file	120
Opening an existing message category file.	121
Adding a message to a message category	121
Deleting a message from a message category	122
Viewing or configuring message category file properties	123
Deleting a message category file	123
Importing	124
Importing file systems into the workbench	124
Importing from C	126
Importing from COBOL copybooks	128
Importing from IBM supplied messages	131
Importing from WSDL	131
Importing from XML DTD	134
Importing from XML schema	136
Generating information from message sets	139
Generating message set documentation as HTML.	139
Generating XML schema	140
Generating a Web Service Definition from a message set	141



---

## Developing message models

This topic area describes the concepts behind message modeling, and the tasks that are involved in working with message models.

If you are unfamiliar with message models, read the topics that describe the concepts; starting with Message modeling. These topics explain why you need to model messages, and describe the message modelling objects that you will use, such as message sets and message definition files.

The WebSphere Message Broker message model is based on XML Schema. For more information on XML Schema, see XML Schema Part 0: Primer.

The tasks that are involved in developing message models are:

- “Working with a message set project” on page 79
- “Working with a message set” on page 80
- “Working with a message definition file” on page 91
- “Working with message model objects” on page 92
- “Creating a multipart message” on page 117
- “Linking from one message definition file to another” on page 118
- “Working with a message category file” on page 120
- “Importing” on page 124
- “Generating information from message sets” on page 139

**Tip:** The workbench provides a set of toolbar icons that invoke wizards that you can use to create many of the resources associated with message models, for example a new message set project. Hover your mouse pointer over a toolbar icon to see its function.

You will notice that the workbench lets you open resource files with other editors. You are advised to use only the workbench to edit resource files associated with message models because this editor correctly validates all changes that you make to these files.

---

## Message modeling

Much of the business world relies on the exchange of information between applications. The information is contained in messages having a defined structure which is known and agreed by the sender and the receiver. You will probably find that your applications use a combination of messages, including those defined by:

- C and COBOL data structures
- industry standards such as SWIFT or EDIFACT
- XML DTD or schema

You can model a wide variety of message formats which can be understood by WebSphere Message Broker message flows. Once the message format is known, the broker can parse an incoming message bit stream and convert it into a logical message tree for manipulation by a message flow. After the message has been processed by the message flow, the broker converts the message tree back into a message bit stream.

This topic area gives an overview of Message modeling:

- “Message modeling concepts”
- “Why model messages?” on page 6
- “Which message domain and format to use?” on page 7
- “The message model” on page 10
- “Physical formats in the MRM domain” on page 39
- “Ways to create message definitions” on page 64
- “Generate model representations” on page 75

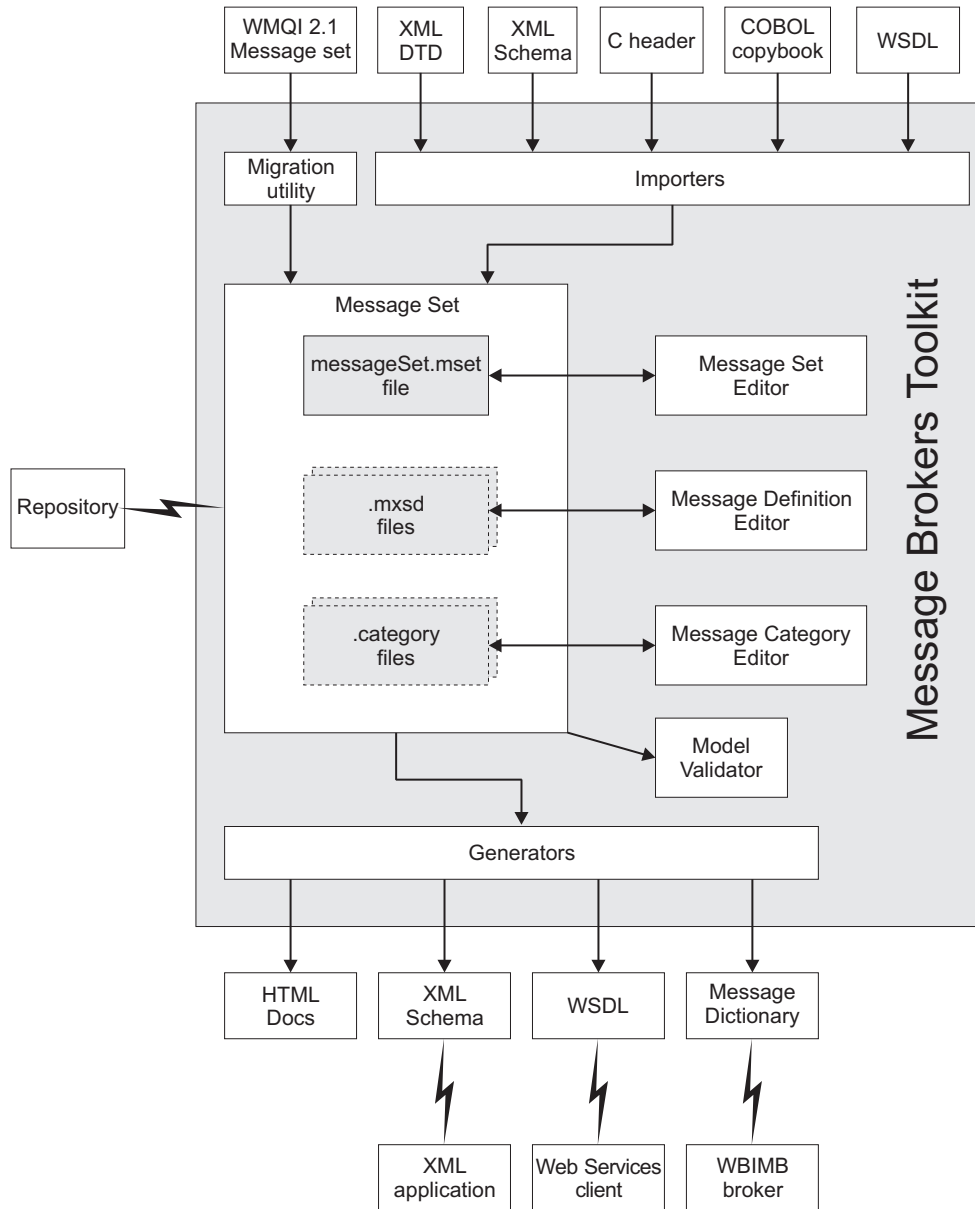
## **Message modeling concepts**

Message modeling is a way of predefining the message formats used by your applications.

Once you have modeled your messages, products like WebSphere Message Broker can use your message models to automatically parse and write your message formats.

When you model messages you need to understand the following concepts:

- Message set projects
- Message sets
- Message definition files
- Message categories
- Model importers
- Model editors
- Model generators
- Model validator
- Domains and parsers



A message set project is a specialized project (container) in which you create and maintain all the resources associated with exactly one message set.

A message set is a logical grouping of your messages and the objects that comprise them (elements, types, groups). The content of a message set is:

- Exactly one message set file
- Zero or more message definition files
- Zero or more message category files

The message set file provides message model information that is common across all the messages in the message set. You can create this information using the message set editor.

Once you have created a message set, you typically import application message formats described by XML DTD, XML Schema, WSDL Files, C structures or COBOL structures, creating and populating message definition files. You can then edit the logical structure of your messages, and create and edit physical formats that describe the precise appearance of your message bit stream during transmission, using the message definition editor. Alternatively, you can create an empty message definition file and create your messages using just the editor.

When your message definition files are complete, you can then generate the message set in a form that can be used by an application. This could be as:

- A message dictionary for deployment to a broker
- An XML Schema for use by an application to validate XML messages for a particular physical format
- Web Services Description Language (WSDL) for a web services client
- Documentation (as HTML) to give to programmers or business analysts

Messages can be optionally grouped into message categories, both for convenience and to aid the generation of WSDL. You can add messages to message categories using the message category editor.

Each time you save a message set file, message definition file or message category file, the content is validated to ensure that the message model you are creating follows certain rules. There are rules for both the logical structure and the physical formats. The validation ensures the integrity of your model, but does not necessarily prevent you saving an invalid message model file.

A product such as WebSphere Message Broker typically supplies a range of parsers to parse and write message formats. Each parser is suited to a particular class of messages (fixed-length binary, delimited text, XML and so on) known as a message domain. Each message set you create specifies a domain, which determines the parser that is used when parsing and writing messages defined within that message set.

## Why model messages?

A product such as WebSphere Message Broker typically supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. An example of a self-defining message format is XML. In XML the message itself contains metadata as well as data values, enabling an XML parser to understand an XML message even if no model is available. Most message formats, however, are not self-defining. As examples, a binary message originating from a COBOL program and a SWIFT formatted text message do not contain sufficient metadata to enable a parser to understand the message. The parser must have access to a model that describes the message to parse it correctly.

Even if your messages are self-defining and do not require modeling, there are still advantages in modeling them:

- Enhanced parsing of XML messages. Although XML is self-defining, without a model all data values are treated as strings. If a model is used then the parser knows the data type of data values and can cast the data accordingly.
- Improved productivity when writing ESQL. When you are creating ESQL programs for WebSphere Message Broker message flows, the ESQL editor can use message models to provide code completion assistance.

- Drag-and-drop message maps. When you are creating message maps for WebSphere Message Broker message flows, the Mapping editor uses the message model to populate its source and target views. Without message models you can not use the Mapping editor.
- Runtime validation of messages. Without a model it is not possible for a parser to check that input and output messages have the correct structure and data values.
- Reuse of message models in whole or in part by creating new messages based on existing messages.
- Automatic generation of documentation.
- Provision of version control and access control for message models by storing them in a central repository.

To make full use of the facilities offered by WebSphere Message Broker, it is recommended that you model your message formats.

To speed up creation of message models, importers are provided that take metadata such as C header files, COBOL copybooks, XML Schema and DTDs, and WSDL files, and create message models from that metadata. Alternatively, IBM has pre-built models for common industry standard message formats such as SWIFT, EDIFACT, X12, FIX, HL7 and TLOG.

## Which message domain and format to use?

Each message set you create specifies a domain, which determines the parser that is used when parsing and writing all messages defined within that message set. Each domain is suited to a particular class of messages (fixed-length binary, delimited text, XML and so on). Additionally, some domains support several physical formats (sometimes called wire formats). When you create a message set you need to decide which domain and physical format to use.

You can select one of the following domains:

**MRM** For modeling a wide range of messages including XML, fixed-format binary and formatted text.

**XML, XMLNS or XMLNSC**

For messages conforming to the W3C XML standard.

**JMSMap or JMSStream**

For messages produced by the WebSphere MQ implementation of the Java Messaging Service standard.

**IDOC** For messages in SAP IDoc format.

**MIME** For handling multipart MIME messages such as SOAP with Attachments or RosettaNet.

Choose the domain which best fits your messaging needs:

- If your messages are in XML, use either the XML Wire Format in the MRM domain, or use the XML, XMLNS or XMLNSC domains. Usually, you will find that the MRM domain offers more flexibility. See “XML messages in the MRM and XML domains” on page 10 for more details about the relative strengths and weaknesses of these options.
- If your messages come from a legacy application written in a language such as C, COBOL or PL/1, or consist of fixed-format binary data (possibly including null-terminated strings), use the Custom Wire Format in the MRM domain.

- If your messages consist of formatted text, perhaps with field content identified by tags or separated by specific delimiters or both, use the Tagged Delimited String Format in the MRM domain.
- If you use JMS messages, use either the XML domain, or one of the JMS domains.

Alternatively, you can specify your own domain. You would typically do this if you were using WebSphere Message Broker and had written a custom plug-in parser to parse your messages, instead of using the MRM or XML parsers.

## The MRM domain

The MRM domain can be used for modeling a wide range of messages.

The key features of the MRM domain are:

- Support for modeling messages from legacy applications written in C, COBOL, PL/1 and other languages, using the Custom Wire Format (CWF) physical format. This support includes the ability to create a message model directly from a C header file or COBOL copybook.
- Support for modeling XML messages, including those that exploit XML namespaces, using the XML Wire Format (XML) physical format. This support includes the ability to create a message model directly from an XML DTD or XML Schema file.
- Support for modeling formatted text messages, perhaps with field content identified by tags or separated by specific delimiters or both, using the Tagged Delimited String Format (TDS) physical format. This includes industry standards such as SWIFT, EDIFACT and X12.
- Easy transformation from one physical format to another. For example, a model can be created by importing a C header file, and the equivalent XML Schema can be generated for use by a different application.
- The MRM domain comes with a specialized parser that is used by runtime products such as WebSphere Message Broker. Once you have completed your message set, you generate a message dictionary which is deployed to the runtime product. The MRM parser then uses the dictionary to parse and write messages that are defined within that message set.
- The MRM parser can perform runtime validation of messages against the deployed dictionary. See “XML messages in the MRM and XML domains” on page 10 for more details about the relative strengths and weaknesses of using the MRM domain or one of the XML domains to model XML messages.

## The XML domains

The XML domains can be used for modeling messages conforming to the W3C XML standard.

Each of the three XML domains can be used to model W3C XML messages.

### XMLNSC domain

This domain may be used for any XML message, including those that use XML namespaces. The accompanying parser reduces the amount of memory taken up by the logical message tree created from the parsed message. Its default behavior is to discard non-significant whitespace and mixed content, comments, processing instructions and embedded DTDs, though controls are provided to retain mixed content, comments and processing instructions if desired.



### **XMLNS domain**

If the XMLNSC domain does not meet your requirements, use this namespace-aware domain.

### **XML domain**

This domain is not namespace aware and is provided for compatibility with earlier releases of WebSphere Message Broker.

The MRM domain is also capable of processing XML messages, and there are significant advantages in using the MRM domain for XML. See “XML messages in the MRM and XML domains” on page 10 for more information.

Note that it is not essential to model messages that are processed by the XML domains. WebSphere Message Broker parses such messages without reference to any message model. However, there are several advantages in modeling such messages, which are described in the “Why model messages?” on page 6 topic area

### **The JMS domains**

The JMSMap and JMSStream domains can be used for modeling messages produced by the WebSphere MQ implementation of the Java Messaging Service standard.

These message types map naturally to the XML domain, and are therefore supported in an identical fashion to XML domain messages.

The XML domain supports messages conforming to the W3C XML standard that do not exploit XML namespaces. If your JMS messages exploit XML namespaces you should use the XMLNS or XMLNSC domain instead.

Alternatively, you can define JMS messages in the MRM domain if you choose, and so take advantage of the extra features of the MRM domain such as message validation.

### **The IDOC domain**

The IDoc domain processes messages that SAP R3 clients send to the broker across the MQSeries link for R3.

A SAP IDoc message contains both SAP-defined and user-defined fields. The parser supplied with the IDOC domain creates a logical message tree from the SAP-defined fields, then automatically invokes the MRM domain parser to complete the tree from the user-defined fields. It is therefore necessary to model the user-defined content of the IDoc message using a message set that contains a Custom Wire Format (CWF) physical format.

One of the message definitions supplied with WebSphere Message Broker is a model of the SAP-defined fields. The model can be added to a message set using the New Message Definition file wizard. This enables other editors to make use of the SAP-defined fields, for example, the WebSphere Message Broker ESQL editor uses the model to provide code completion.

### **The MIME domain**

The MIME domain implemented in WebSphere Message Broker does not support the full MIME standard, but does support the MIME formats in use in message based applications, including SOAP with Attachments and RosettaNet.

The MIME domain should be used if your messages exploit the MIME standard for multipart messages. The MIME parser will create a logical tree in which each

part of your original message is held as a separate BLOB. The individual parts may then be further parsed as required using ESQL or a ResetContentDescriptor node.

Note that the MIME parser is not a model-driven parser, but relies entirely on the bitstream metadata to create its logical tree. Similarly when a MIME tree is written back to a bitstream, the MIME domain relies on the structure and metadata of the tree itself. This makes it particularly important to understand the mapping between the bitstream and the logical tree, and its relationship to the ContentType property.

A model of the MIME multipart message structure is available as an IBM supplied message definition, and may be added to a message set using the New Message Definition file wizard. This enables features in other editors, such as the WebSphere Message Broker ESQL editor code completion feature, to make use of MIME header fields.

### **XML messages in the MRM and XML domains**

If your messages are in XML, you can use either the XML Wire Format in the MRM domain, or one of the XML domains (XMLNSC, XMLNS or XML).

You are recommended to model your XML messages using the MRM domain, for the following reasons:

- You can generate a message dictionary for use by the MRM parser in WebSphere Message Broker. This enables the MRM parser to interpret the data in an XML message in an advanced manner.

For example:

- The dictionary indicates the real data type of a field in the message, instead of always treating it as a character string.
- The MRM parser can validate XML messages against the dictionary.
- Base64 binary data can be automatically decoded.
- Date and time information can be extracted from a data value using a specified format string.

- You have extra control over the rendering of your data.

For example, you might have a data field that is rendered as an XML element in one message, and as an XML attribute in another message. Or you could have a data field that is known by a particular name in one message, and a different name in another message. This advanced rendering can be specified using MRM XML Wire Format properties.

- You can share a common logical message structure between physical formats. For example, if you have a message that is created by a legacy COBOL application, you can use an MRM XML Wire Format to easily and quickly define the equivalent XML message.

## **The message model**

The message model consists of the following components:

- Message set projects
- Message sets
- Message definition files
- Message categories

See “Message modeling concepts” on page 4 for a summary of these components and the relationship between them. See Related concepts below for a detailed description of each component.

The majority of your model content is described by message definition files. These files use XML Schema to represent your messages. XML Schema is an international standard which defines a language for describing the structure of XML documents. It is ideally suited to describing the messages which flow between business applications, and it is widely used in the business community for this purpose.

Each message definition file describes both the *logical structure* of your messages, and the *physical format(s)* that describes the appearance of your message bit stream during transmission. If you are using the MRM domain, you *must* provide physical format information. This tells the MRM parser exactly how to parse the message bit stream. If you are using one of the XML domains, physical format information is not needed

See “Ways to create message definitions” on page 64 to understand the different ways that you create and populate message definition files. See “Physical formats in the MRM domain” on page 39 for a description of the physical formats available to you.

## Message set projects

A message set project is a specialized container in which you create and maintain all the resources associated with one message set.

The content of a message set project is a single message set folder, the name of which provides the name of the message set. A message set project is created in one of three ways:

- Using the New Message Set Project wizard.
- Using the `mqsimigratemsgsets` command line utility. A new message set project is created for each migrated message set.
- Using the `mqsicreatemsgdefs` command line utility. A new message set project is created if the project identified by the `-p` parameter does not exist.

A message set project has some restrictions that apply to it:

- A message set project must contain zero or one message sets.
- A message set project cannot refer to any other message set.

## Message sets

A message set is a container for grouping messages and associated message resources (elements, types, groups).

A message set is a folder in a message set project that contains a `messageSet.mset` file. The name of the folder is the name of the message set. A message set project can contain just one message set.

A message set is created whenever you create a new message set project. If you already have an empty message set project, you can create a message set in it using the New Message Set wizard.

You can base your new message set on an existing message set. If you do so, all the definitions in the existing message set are copied into the new message set.

When you have created your message set, you must specify two key properties:

### Runtime Parser

The message domain of the message set. This is used when parsing and writing all messages defined within that message set and effectively defines the message domain.

### Use Namespaces

Indicates whether the message definitions you create within the message set are XML namespace aware.

**Message set resources:** Resources within a message set are created as files, and are displayed under the message set folder in the Navigator view:

- Message set file `messageSet.mset`

There is always one, and only one, `messageSet.mset` file in a message set. It contains message model properties that are common to all the content of the message set. It is also where you define the physical formats that you want for this message set. These can be Custom Wire Format (CWF), XML Wire Format (XML) and Tagged Delimited String Format (TDS).

The file is created for you when a new message set is created, and you manipulate its content with the Message Set Editor.

- Message definition files that have the suffix `.mxsd`

You can have many message definition files in a message set. Each file contains the logical model and associated physical model in XML Schema form for a group of related messages.

- Message category files that have the suffix `.category`

These files are optional. You can have many message category files in a message set. A message category provides another way of grouping your messages, perhaps for documentation purposes or to assist with generating Web Services Description Language (WSDL) files.

When you have completed the resources in your message set, you can generate the message set in a form that can be used by an application. This could be:

- a message dictionary for deployment to a broker
- XML Schema for use by an application building XML messages
- Web Services Description Language (WSDL) for a web services client
- documentation (HTML) to give to programmers or business analysts

**Message set identification:** As well as having a name, provided by the folder, and a message set alias, a message set also has an identifier. This is a generated string of 13 characters that uniquely identifies the message set. You can use either the name, the alias, or the identifier interchangeably whenever you need to specify the message set to a runtime product such as WebSphere Message Broker (for example, in an MQRFH2 message header).

Although the name and alias are more readable, the identifier is guaranteed to be unique and is intended for use where you do not have control over the names of the message sets you are using.

**Message set recommendations:** You can have as many message definition files as you want within one message set. But you are recommended to limit your message sets to a few related message definition files that share the same physical formats. There are several reasons for this:

- Generation of a message dictionary and other representations is quicker.
- Generated documentation is more manageable.

- Physical formats apply to all objects within the message set.  
So for example, if you have an XML message and an unrelated CWF message in the same message set, then both XML and CWF physical format properties will be present for all objects. But the XML properties are of no interest to the CWF message and so will take default values in that message. (Similarly, the CWF properties are not of interest to the XML message.) This can result in unwanted task list warnings.
- Recursion is only permitted for the XML physical format, not for CWF and TDS.  
So if you are modeling XML messages that have a recursive structure, you must ensure that recursive XML messages do not share a message set with non-XML physical formats.

**IBM Supplied Message Sets:** A collection of predefined, read-only message sets that model the headers used by WebSphere MQ messages are supplied. These are useful if you are referencing fields in these headers in WebSphere Message Broker message flows.

MQWIH	WebSphere MQ work information header
MQCIH	WebSphere MQ CICS bridge header
MQRFH	WebSphere MQ rules and formatting header
MQRFH2	WebSphere MQ rules and formatting header - version 2
MQDLH	WebSphere MQ dead letter header
MQSAPH	WebSphere MQ link for R/3 header
MQMD	WebSphere MQ message descriptor
MQIIH	WebSphere MQ IMS information header
SMQ_BMH	WebSphere MQ link for R/3 bad message header
MQMDE	WebSphere MQ message descriptor extension
MQRMH	WebSphere MQ message reference header
MQCFH	WebSphere MQ programmable command formats header

**Message set version and keywords:** When you are developing a message set, you can define the version of the message set as well as any other key information that you need to be associated to it. After the message set has been deployed, the Configuration Manager can be used to display the properties of the message set. This includes the deployment and modification dates and times (the default information that is displayed) as well as any additional version or keyword information that you have set.

This allows you to define information to give details on the message set that has been deployed, therefore you can check that it is the message set that you expect.

## Version

You can set the version of the message set in the Version property. This is in the Documentation view of the message set.

You can also define a default message set version in the Default version tag of the message set preferences. All new message sets that are created after this has been set have this default applied to the Version property at the message set level.

## Keywords

Keywords need to be defined in the Documentation property of the message set. These follow certain rules to ensure that the information can be parsed. The following is an example of what you can define in the Documentation property:

```
$MQSI Author=John Smith MQSI$  
$MQSI Subflow 1 Version=v1.3.2 MQSI$
```

The information the Configuration Manager will show is:

Message set name	
Deployment Time	28-Aug-2004 15:04
Modification Time	28-Aug-2004 14:27
Version	v1.0
Author	John Smith
Subflow 1 Version	v1.3.2

In this display the version information has also been defined using the Version property of the object. If the version information has not been defined using the property, it is omitted from this display.

## Message definition files

A message definition file contains the messages, elements, types and groups which make up a message set.

Every message set requires at least one message definition file to describe its messages. Message definition files use the XML Schema language to describe the logical format of one or more messages. Extra information in the form of XML Schema annotations is used to describe any physical formats which you define for the messages.

Large message sets can contain several message definition files. This keeps the individual files to a manageable size, making them faster and easier to work with.

Message definition files can be created using the Message Definition Editor, or can be imported from a range of different file formats as described in the topic, "Import from other model representations to create message definitions" on page 64.

A message definition file can be associated with a namespace, so that all message model objects declared within the file belong to that namespace. Namespaces provide a means of avoiding name clashes among similarly named global objects. They are described in detail in the topic, "Namespaces" on page 35.

One message definition file can reuse message model objects defined in another message definition file. XML Schema provides two mechanisms to do this: import and include. For more information see "Reusing message definition files" on page 38.

**XML Schema:** XML Schema is an international standard which defines a language for describing the structure of XML documents. The XML Schema language is ideally suited to describing the messages which flow between business applications, and it is widely used in the business community for this purpose.

WebSphere Message Broker uses XML Schema to describe the logical structure of messages. At a simple level, the types and elements in the message are modeled using XML Schema types and elements. However, when the need arises, most of the advanced modeling features of XML Schema are available for modeling MRM messages.

Some important restrictions and extensions of XML Schema are applied in order to remain compatible with previous releases of the product. These are documented in "Schema restrictions" and "Schema extensions."

*Further information about XML Schema:* For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

*Schema restrictions:* This topic provides information on XML Schema features which are not supported in the message model. Most of these differences are necessary to maintain compatibility with previous versions of the product, and should not affect most users. Future versions of the product will add support for some of these features.

*Unsupported XML Schema features:* The following features are accepted, but not supported, and will give rise to validation errors if they are used in your message model:

- Redefines

The following constructs are accepted, but ignored and will have no effect if deployed to WebSphere Message Broker:

- Identity constraints (unique, key, keyref)
- Whitespace facets
- Pattern facets on simple types other than xsd:string

*Further information about XML Schema:* For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

*Schema extensions:* This topic provides information on areas where the message model provides facilities which are beyond the XML Schema specification.

*Messages:* A message is a global element representing an entire message (rather than a structure within a message). Within a message definition file, a message is represented by a special global element which carries the extra information required by WebSphere Message Broker.

*Composition:* The message model adds the following compositions which are beyond the XML Schema specification. These extra compositions allow message models produced in earlier versions of the product to be supported.

#### **orderedSet**

A set of elements which must appear in the order in which they are listed. Groups are not allowed within an orderedSet. Elements can repeat, but duplicate element references are not allowed.

#### **unorderedSet**

A set of elements which can appear in any order. Groups are not allowed

within an `orderedSet`. Unlike an `all` group, elements within an `orderedSet` are allowed to repeat. However, duplicate element references are not allowed.

#### **message**

A refinement of choice which is only allowed to contain a set of references to messages within the same message set. Groups and complex types with composition of message are used when modeling multipart messages.

*Physical format information:* If one or more physical formats are defined for a message set, the XML Schema objects within the message set can hold extra information about how they should be serialized.

*Further information about XML Schema:* For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

**Message model objects:** This is an introduction to the various objects which make up a message model.

#### **Message**

A message is a set of data passed between applications. Messages must have a structure and format which is agreed by the sending and receiving applications.

#### **Message definition**

A message definition is a logical description of a message. A message definition is a collection of simple elements organized in a tree-like structure.

#### **Simple element**

A simple element describes one or more fields in a message. It is based on a simple type (for example, string, integer or float). It can repeat, and it can define a default or a fixed value.

#### **Simple type**

A simple type describes a class of data within a message. It describes the type of data (for example, string, integer or float) and it can have value constraints which place limits on the values of any simple elements based on that simple type.

#### **Complex element**

A complex element is a named structure containing simple elements within the message. Complex elements can contain other complex elements, and can also contain groups. The content of a complex element is defined by a complex type.

#### **Complex type**

A complex type describes a structure within a message. It contains elements, attributes and groups organised into a hierarchy.

**Group** A group is a list of elements with information about how those elements can appear in a message. Groups can be ordered (sequence) unordered (all) or selective (choice).

#### **Attribute**

An attribute represents an XML Schema attribute. Attributes are very similar to simple elements, but they require special treatment when used with XML messages. In messages which are not XML, they are treated exactly like a simple element based on the same simple type.



*Global and local objects:* Many of the objects in the message model can be either global or local. A global object must have a unique name, which is used to refer to the object from one or more places in the message model. Local objects are defined and used in only one place in the message model.

You are recommended to make objects local unless they need to be used in more than one place. This reduces the probability of name clashes among the global objects in the message model, and will make the message set easier to work with.

*Properties of message model objects:* The properties of all message model objects are listed on the 'properties' pane of the message editor. The properties fall into three categories:

### **Logical**

The logical properties of an object relate to the format-independent description of the object called the 'logical model'. Logical properties describe what data the object contains without saying anything about how it is written down.

### **Physical**

The physical properties of an object describe how the object is written down. These properties control the parsing and writing of the object. There is one set of physical properties for each physical format in your message set.

### **Documentation**

This field is present for all message model objects. It provides a standard place for any description of the object which you might require. Text entered here does not affect the processing of messages in any way.

*Messages:* A message is a set of data passed from one application to another. A message can be modelled by a message definition which describes the structure and content of the message.

A message definition consists of elements organized into a logical structure agreed by the sending and receiving applications. This logical structure can be modeled using the Message Definition editor, so that messages can be parsed into a logical tree and manipulated easily.

In the message model, a message is always based on a global element. The global element's type describes the contents of the global element, and therefore describes all of the content of the message.

*Multipart messages:* If necessary, a message definition can contain other message definitions. This is necessary for modeling certain large and complex messaging standards like SWIFT and EDIFACT.

*Message identification:* Messages are identified by their name or an alias. The alias is an optional user-specified string to identify the message. The name or alias of a message must be unique within a message set.

*XML schema model:* In the message definition file, a message is modeled as an XML schema element declaration. Extra information is provided by XML schema annotations on the element declaration.

*Elements:* An element is a named piece of information, or a field, within a message, with a meaning agreed by the applications that create and process the message.

An element has a specific meaning agreed by the applications that create and process the message. For example, a message might include a string that your applications have agreed is a 'Customer Name'. An element is always based on a type, either simple or complex.

An element:

- Has a business meaning.
- Is an instantiation of a simple or complex type.
- Can be accessed by name from ESQL in a message flow node.
- Is further defined by its type. For example, the type defines the range of values an element can have.
- Can be defined globally or locally. See related information for more details.

*Simple and complex elements:* Elements can be simple or complex. A simple element is a single, named piece of information such as 'Age' or 'Customer Name'. A simple element is based on a simple type which defines its content.

A complex element is a named structure containing other elements. A complex element named 'Customer Details' might contain the simple elements 'Age' and 'Customer Name'. A complex element can also contain other complex elements. A complex element is based on a complex type, which defines its content and structure.

*Global and local elements:* Elements can be global or local. A global element can be used in several different messages, or even in several places within the same message. It must be given a unique name by which it can be referenced. A local element is defined in one position within one complex type or group, and is not available for reuse elsewhere in the message model.

*Optional and repeating elements:* Elements can be defined as optional, mandatory and repeating using the properties Min Occurs and Max Occurs. For further information, see "Cardinality: optional, repeating and mandatory elements" on page 32.

*Default and fixed values:* An element can be given a default value, so that if no value is supplied by the input message, the default will be used. Alternatively, a fixed value can be defined, and the element will always take that value.

*Value constraints:* An element's value can be constrained using value constraints which define the range of legal values for the element. The value constraints are actually associated with the simple type on which the element is based. For further information, see "Simple types" on page 20.

*Defining substitution groups:* An element can be marked as a valid substitute for another element by using the substitution group property on the element. In this way groups of elements can be assembled where any of the elements in the group can substitute for one element, the head element. For further information, see "Substitution groups" on page 33.

*XML Schema model:* In the message definition file, an element is modeled using an XML Schema element declaration, as described in XML Schema Part 0: Primer. Fixed and default values are modeled using a value constraint on the element. Other types of constraint are applied by using facets on the element's type.

*Further information about XML Schema:* For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

*Wildcard elements:* A wildcard element represents an element which does not appear in the message model, but which could appear at the same position as the wildcard element in the input message.

Wildcard elements provide a means of adding flexibility to the message model, so that messages can be parsed even if they do not exactly match the message model.

Wildcard elements can only appear within a complex type or group with Composition of sequence and Content Validation of closed. Wildcard elements provide a similar capability to setting the Content Validation property of a complex type or group to Open or Open Defined.

### **WebSphere Message Broker**

If you generate a dictionary from your message set and deploy it to WebSphere Message Broker, wildcard elements will be validated against the model according to the following rules:

- If Process Content is set to strict, only elements which are declared in the same message set will be allowed to appear in place of the wildcard element.
- If Process Content is set to lax or skip, any element will be allowed to appear in place of the wildcard element.

If the broker is prior to WebSphere Message Broker Version 6.0, the number of elements permitted to match against the wildcard element is unpredictable (Min Occurs and Max Occurs are ignored).

*Types:* Types describe the data content of elements.

Simple types describe simple elements with data types such as string, integer or datetime.

Complex types describe complex elements - elements which contain a hierarchy of other elements.

For more information, see:

- "Simple types" on page 20
- "Complex types"

*Complex types:* A complex type describes the structure of one or more complex elements.

Complex types are an essential part of every message model, because they define the logical structure of the messages and elements in the model.

*What is a complex type for?:* Complex types define the structure of the messages and elements in the message model. By combining elements, attributes, groups and wildcards, almost any message structure can be modeled.

*Contents of a complex type:*

### **Elements**

Most complex types contain some elements, and some contain a large hierarchy of complex elements. The elements within a complex type are

always contained within a group. This group can be local to the complex type, in which case the Message Definition Editor hides it from view. This is the usual case.

Alternatively, the group containing the elements can be a global group, and this group defines the element content, the composition and the content validation for the complex type.

If a complex type is derived from a simple type, it is not allowed to contain any element content.

### **Attributes**

If you are modeling XML messages, your complex types can contain attributes. The attributes for a complex type can be local or global, and they can be contained within an attribute group.

### **Groups**

Groups allow sets of elements to be included in a complex type. The members of the group are included as peers of the other elements. For more information about their use, see “Groups” on page 23.

### **Wildcards**

Complex types can contain wildcard elements, which allow unmodeled elements to appear within any elements based on the complex type. Any such elements must appear at the same position within the message as the wildcard. Complex types can also contain wildcard attributes, which allow unmodeled attributes to appear within any elements based on the complex type.

*Composition:* The composition of a complex type describes how the members of the type are organized. For more information, see “Groups” on page 23.

*Controlling validation of type content:* The Content validation parameter on a complex type specifies how strictly the contents of the type should be validated. For more information, see “Groups” on page 23.

*Substitution Settings:* A complex type has parameters which control whether other types can be derived from it (final) and whether other types can substitute for it (block). For more information, see “Substitution groups” on page 33.

*Simple types:* A simple type is an abstract definition of an item of data such as a number, a string or a date.

The purpose of a simple type is to define the content of one or more simple elements. Simple types (and any elements based on them) cannot contain attributes or child elements. Simple types stand in contrast to complex types, which define the structure of an element but usually do not define any simple data.

## **Global and local simple types**

Simple types can be global or local. A global simple type can be used as the basis for more than one element. It must be given a unique name by which it can be referenced. A local simple type is associated with a single element, and is not available for reuse elsewhere in the message model. Local types do not need a name, and are displayed as {Local simpleType} by the message definition editor.

## **Variations of simple types**

### **Restriction**

A simple type can be derived from another simple type (the base type) by restriction. A restriction type can have value constraints applied to it.

A restriction type may be defined as a built-in simple type, a user defined simple type or a local simple type defined anonymously within the restriction.

## List

A list type is a way of rendering a repeating simple value. The notation is more compact than the notation for a repeating element, and offers a way to have multi-valued attributes.

A list type can be based on a union type. This can describe a space-separated list of items in which each item can be based on any of the simple types in the union.

A list of lists is not legal. The item type of a list cannot be a list itself or derived at any level from another list type and will result in a task list error in the editor.

A list type can have the facets of `minLength`, `maxLength` and `length` applied to it. These facets restrict the number of items in the list. To restrict the values of each item in the list, facets should be applied to the item type and not the list itself. The message definition editor provides additional support for enumeration and pattern facets directly on a List type, to enable the import of any schema using them, but will issue a warning that enumeration and pattern facets will be ignored by the Broker.

## Union

A union type is as a union of two or more other simple types.

A union type allows a value to conform to any one of several different simple types. The simple types that compose a union type are known as its member types. There is no upper limit on how many member types may exist, but there must be at least one. A member type may be defined as a built-in simple type, a user defined simple type or a local simple type defined anonymously within the union type.

A union type can also include list, union and restricted simple types among its members.

The broker does not apply value constraints until the data is in the logical tree. This means it will not be possible to choose between two simple types which are derived from the same fundamental type, but with different constraints (for example, integer with range 1-10 and integer with range 11-20). A warning will appear in the task list if this is attempted and the broker will simply ignore the constraints when resolving the union. The message definition editor provides additional support for enumeration and pattern facets directly on a Union type, to enable the import of any schema using them, but will issue a warning that enumeration and pattern facets will be ignored by the broker.

## Value constraints

Any value constraints applied to the derived type must further restrict the base type (and any elements based on it). It is illegal for a derived type to weaken or remove a value constraint which its base type has defined. If no value constraints are applied to the derived type, it will be almost identical to its base type, although it will be treated as a restriction of the base type in situations where that

is relevant (type inheritance and element substitution).

*Type inheritance:* The XML Schema language allows a type definition to be based on another type definition. In this way, a hierarchy of types can be constructed. This topic outlines the concepts of type inheritance, and highlights some important issues relating to substitution.

A full discussion of XML Schema type inheritance can be found on the World Wide Web Consortium (W3C) Web site, or in numerous books on XML schema.

*Restriction and extension:* A type is a restriction of its base type if elements of the derived type have a smaller range of legal values (or legal type members) than elements of the base type.

- For example, a restriction of a complex type might reduce the number of occurrences of one of its type members, or might omit that type member completely.
- Similarly, a restriction of a simple type might lower the Max Inclusive facet value, or raise the Min Inclusive facet value.

A type is an extension of its base type if elements of the derived type have a wider range of legal values (or legal type members) than elements of the base type.

- For example, an extension of a complex type might add type members which were not present in the base type, or might allow a type member to repeat.
- Similarly, an extension of a simple type must always be a complex type which is based on the simple type. (It is not possible to extend a simple type by widening its range of legal values)

Special rules apply to the derivation of simple types. A simple type cannot extend another simple type. This ensures that restrictions imposed by a simple type cannot be removed by deriving another simple type from it.

However, a complex type is allowed to extend a simple type. This does not affect the range of legal values of the simple type, but it does allow attributes to be added. The result of extending a simple type is always a complex type containing zero or more attributes.

*Controlling type inheritance:* The final attribute on a complex type can take three values, with the following effects:

- restriction: It is illegal to derive another complex type from this type by restriction.
- extension: It is illegal to derive another complex type from this type by extension.
- all: It is illegal to derive another complex type from this type by either extension or restriction

*Type inheritance and substitution:* XML Schema provides two different substitution mechanisms, both of which use type inheritance information to allow or disallow substitutions.

Element substitution is controlled by substitution groups, and element substitution can be blocked or allowed for extension and restriction by settings on either the element itself, or on the element's type.

Type substitution allows the type of the element to be defined within the instance document, using the `xsi:type` attribute on the element, so that the element's real

type is not known until the element has been partly parsed. This mechanism can also be blocked or allowed based on the derivation method of the types involved.

*Groups:* A group is a list of elements which defines how those elements can appear in a message.

Groups can be ordered (sequence) unordered (all) or selective (choice). Groups define the composition and content validation of a set of type members.

*What are groups for?:* Groups can be used for any of the following purposes:

- To define the entire content of a complex type.  
A complex type can refer to a global group which completely defines its content. (If it does not, the content of the complex type is defined by an anonymous local group, which is hidden within the Message Definition Editor.)
- To represent a common sub-structure within more than one type.  
Two or more complex types can refer to the same global group if they both contain the same subset of elements.
- To change the composition midway through a complex type.  
You might have a complex type which is a sequence of three members, but the second member is a choice of two elements. To model this, a group with composition set to choice can be inserted as the second member of the sequence.

*Contents of a group:* Groups can contain complex elements, simple elements, wildcard elements and groups.

By combining these components, the structure of any message can be modeled. Wildcard elements can be included to allow unmodeled elements to appear, thus making the message model robust and flexible.

*Global and local groups:* Groups can be global or local.

A global group can be used in more than one place in the message model. It represents a structure which appears in more than one place in the message model. A global group must be given a unique name by which it can be referenced.

A local group is defined in one position within one group, and is not available for reuse elsewhere in the message model. Local groups do not need to be named.

In XML Schema, a group can have its composition set to sequence, all or choice.

- A sequence is simply a set of elements which must appear in same order as they are listed.
- An all group is a set of elements which can appear in any order, and cannot repeat.
- A choice is a set of elements, only one of which can appear in any given message.

The message model also allows other compositions, in order to remain compatible with earlier versions of the product. For further information, see “Schema extensions” on page 15.

*Controlling validation of type content:* The Content validation property on a group determines how strictly the content of the group should be validated. If Content validation is set to:

- Closed, the contents of the group are validated strictly against the model

- Open Defined, elements which are declared within the same message set are allowed to appear as children of the group, even if they are not modeled
- Open, no validation will be performed

*Attributes:* An attribute describes an XML attribute, or any simple element within a message.

Attributes are provided to simplify the modeling of XML messages - if none of your messages use the XML physical format, you should consider using simple elements instead.

*Attributes and XML:* The most common use for an attribute is to model an XML attribute within an XML message. In this scenario, each attribute which can appear in the XML message has a corresponding attribute in the logical message definition.

*Attributes in other physical formats:* Sometimes a message needs to be parsed as XML but written in another physical format (Custom Wire Format or Tagged Delimited String Format). In this case, any attributes in the message will be treated exactly like simple elements with the same properties.

*Global and local attributes:* Attributes can be global or local.

A global attribute can be used in more than one place in the message model. It must be given a unique name by which it can be referenced.

A local attribute is defined in one position within one complex type, and is not available for reuse elsewhere in the message model.

*Optional and repeating elements:* Attributes can be defined as optional, required or prohibited. Attributes are not allowed to repeat. For further information, see “Cardinality: optional, repeating and mandatory elements” on page 32.

*Default and fixed values:* An attribute can be given a default value, so that if the attribute is missing from the input message, the default will be used. Alternatively, a fixed value can be defined, and the attribute will always take that value.

*Value constraints:* An attribute’s value can be constrained by using value constraints which define the range of legal values for the attribute. The value constraints are actually associated with the simple type on which the attribute is based. For further details, see “Simple types” on page 20.

*XML Schema Model:* In the message definition file, an attribute is modeled as an XML schema attribute declaration. Fixed and default values are modeled using a value constraint on the attribute. Value constraints are modeled as facets on the attribute’s simple type.

*Wildcard attributes:* A wildcard attribute allows unmodeled attributes to appear in a message.

The Process Content and Namespace properties control the namespace to which the unmodeled attributes must belong.

### **WebSphere Message Broker**

If you generate a dictionary from your message set and deploy it to WebSphere Message Broker, wildcard attributes will be validated against the model according to the following rules:



- If Process Content is set to strict, only attributes which are declared in the same message set will be allowed to appear in place of the wildcard attribute.
- If Process Content is set to lax or skip, any attribute will be allowed to appear in place of the wildcard attribute.

**Tip:** If the namespace property is set to the namespace of the message set, these rules are then similar to the behaviour of a validating XML parser.

*Attribute groups:* An attribute group defines a set of attributes which can appear in a complex type.

An attribute group provides a way to include the same set of attributes in more than one complex type, without duplicating the definitions.

For example, if most of the elements in your message model have the attributes 'amount', 'currency' and 'date', these could be put into a single attribute group, which is referenced by all the complex types which use them.

*Value constraints:* Value constraints refine a simple type by defining limits on the values which it can represent.

It is often useful to be able to constrain the values which an element can take, perhaps to ensure that messages conform to business rules. This topic describes how value constraints can be added to a simple type in order to constrain the values of all elements based on it.

Most of the value constraints discussed here are modeled by XML Schema facets, and are associated with a simple type. However, default values and fixed values can be defined for an element, and these are associated with the element itself.

**Tip:** If the message set is deployed to WebSphere Message Broker, all value constraints can be validated, so that any violations will be reported as errors or warnings.

*Types of value constraint:*

#### **Default Value**

If an empty element is present in an input message, it will be assigned the default value. If an element is completely missing from an input message, it will not be assigned any value at all.

If an attribute is missing from an input message, it will be assigned the default value.

Note that default value constraints are associated with the element itself, and therefore cannot be inherited.

**Note:** If you deploy a message set to WebSphere Message Broker, the behavior described above will not always apply. For details of the differences, refer to the *Relationship to the logical model* topic for the appropriate physical format.

#### **Fixed Value**

If an element appears in an input message, its value must be the specified fixed value. If an empty element is present in an input message, it will be

assigned the fixed value. If an element is completely missing from an input message, it will be assigned the fixed value.

If an attribute is present in an input message, it must have the fixed value. If an attribute is missing from an input message, it will be assigned the fixed value.

Fixed value constraints are associated with the element itself, and therefore cannot be inherited.

**Note:** If you deploy a message set to WebSphere Message Broker, the behavior described above will not always apply. For details of the differences, refer to the *Relationship to the logical model* topic for the appropriate physical format.

### **Length Constraints : Length, Min Length, Max Length**

Using length constraints, the length of all elements based on the simple type can be constrained or even limited to a single value.

Length constraints can be applied to simple types which are derived from `xsd:hexBinary`, `xsd:base64Binary` or `xsd:string` (including built in schema types such as `xsd:normalisedString`).

Length constraints are inherited from ancestor types, and any length constraints defined for a simple type must not relax the constraints imposed by any of its ancestor types. For example, it would be illegal to derive a type 'longString' (Max Length=100) from a type 'shortString' (Max Length=10).

**Note:** For WebSphere Message Broker by default, Length value constraints are converted to Max Length constraints when a message set is added to a BAR file. This avoids WebSphere Message Broker raising spurious validation errors for fixed-length data structures, where the strings tend to be padded to fit a fixed-width field. If strict length validation is required, this default can be changed in the message set properties by changing the flag Treat Length facet as Max Length.

### **Range constraints : Min Inclusive, Max Inclusive, Min Exclusive, Max Exclusive**

Range constraints specify the allowable range of values for all elements which are based on the simple type. Inclusive constraints include the specified endpoints in the allowed range, whereas exclusive constraints do not. Range constraints can be applied to simple types which are numerical or which relate to calendar and time values. They cannot be applied to strings, because the ordering of string values depends on the character set being used.

Range constraints are inherited from ancestor types, and any range constraints defined for a simple type must not relax the constraints imposed by any of its ancestor types. For example, it would be illegal to derive a type 'largeNumber' (Max Inclusive=100) from a type 'smallNumber' (Max Inclusive=10).

The message model does not allow exclusive constraints to be applied to non-integral types (float, decimal, double, dateTime, and so on).

### **Enumeration constraints**

An enumeration constraint specifies a single allowed value for all elements which are based on the simple type. A list of allowed values can be specified by defining more than one enumeration constraint for the same simple type. Enumeration constraints can be applied to all simple types.

Enumeration constraints are inherited from ancestor types, and any set of enumeration constraints defined for a simple type must not increase the range of allowed values. For example, it would be illegal to derive a type 'AllColours' (with enumerations for all colours of the rainbow) from a type 'MonoColours' (with enumerations for 'black' and 'white' only).

#### **Precision constraints : Total Digits and Fraction Digits**

Precision constraints relate only to decimal values. They limit the number of significant digits (total digits) and the number of decimal places (fraction digits) for all elements based on the simple type. Precision constraints can only be applied to simple types which are derived from xsd:decimal.

Precision constraints are inherited from ancestor types, and any precision constraints defined for a simple type must not relax the constraints imposed by any of its ancestor types. For example, it would be illegal to derive a type 'veryPrecise' (Fraction Digits=10) from a type 'notVeryPrecise' (Fraction Digits=1).

#### **Pattern constraints**

A pattern constraint is a regular expression specifying a set of allowed values for all elements based on the simple type. Multiple patterns can be defined for the same simple type, allowing complex validation rules to be expressed in logically separate parts. Each pattern constraint on a simple type contributes to the set of allowed values for elements based on it. In other words, all the patterns are combined using boolean OR.

As with all value constraints, a simple type can inherit pattern constraints from the simple type on which it is based. In this case, the set of pattern constraints contributed by each ancestor type must be satisfied as well as the set contributed by the simple type itself. In other words, the sets of pattern constraints from each level in the type hierarchy are combined using boolean AND.

Pattern constraints can only be applied to simple types which are derived from xsd:string.

#### **Whitespace constraints**

A whitespace constraint specifies how a parser should treat whitespace for all elements based on the simple type.

**Note:** For WebSphere Message Broker, the MRM physical formats allow whitespace to be controlled very precisely for each physical format defined for the message, but these physical properties are separate from the whitespace constraint in the logical model.

**Object identification:** Objects in the message model (elements, attributes, types, groups) are identified by their name only. This means that no two objects in the same scope are allowed to have the same name. Name clashes can be avoided more easily if global objects are used only when necessary. Local objects are not visible outside of the scope of their parent object, so their names can be re-used without causing a name clash.

#### **Namespaces**

If namespaces are enabled for a message set, each message definition file within it can specify a namespace. Namespaces are an XML schema mechanism for organising groups of related objects into a named 'module'.

Global objects in different namespaces are allowed to share the same name, so namespaces offer another means of avoiding name clashes among global objects.

### **Valid names**

Since the message model is based on the XML Schema language, the name of every message model object must be a valid XML Schema identifier. For information on what constitutes a valid XML Schema identifier, see XML Schema Part 0: Primer.

**Note:** For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

**Multipart messages:** A multipart message contains one or more other messages within its structure. The contained message is sometimes referred to as an embedded message.

A multipart message must contain a group or a complex type with its Composition property set to Message. This group or complex type can contain a list of references to messages which are allowed to appear at that location in the message structure, or it may be empty, permitting any message to appear. When a message is parsed, only one message can appear in that position as an embedded message.

An embedded message can be from the same message set as the multipart message, or it can be from a different message set altogether.

*Message envelopes:* A common use of multipart messages is to define an outer message with a fixed structure. This outer message is called the message envelope. Within the message envelope a group or complex type is included, as described above. Examples of message standards that can be modeled using this technique are X12, SWIFT, SOAP XML and SAP IDoc.

*Identifying the embedded message:* When a multipart message is parsed, the parser needs to be able to identify the embedded message (it could be any of the messages referenced by the group or complex type), or it could be a message not referenced by the group or complex type, perhaps from a different message set). This is achieved using one of two techniques, Message Identity or Message Path..

Additionally for XML messages only, the MRM parser automatically identifies embedded messages using the tag in the XML document. In this case, Message Identity and Message Path techniques are not required unless the embedded message resides in a different message set.

*Restrictions:* When a multipart message is received or generated by a broker, all embedded messages must be of the same physical format as the outermost message and have the same character set and encoding.

*Identifying an embedded message using a Message Identity:*

This describes how you can identify an embedded message using the Message Identity. Using the Message Identity to identify an embedded message is the technique that replaces the use of the Message Key.

The Message Identity technique for identifying embedded messages is useful when a multipart message has a format such as that shown in Figure 1 below.

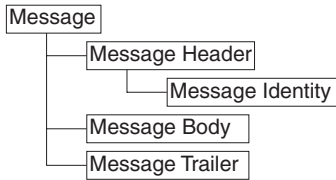


Figure 1. Multipart message structure

In this example, the Message Header and Message Trailer act as an envelope for the message body. They typically will have a fixed structure, although the Message Body can be defined with many different structures.

A place holder for an embedded message is created by setting the Composition property of the complex type or group of the Message Body element to Message. This allows an embedded message to be added at this point within the outer Message, thus creating a multipart message.

When using the Message Identity technique to parse such a multipart message, the embedded message must be identified earlier in the Message Header using a Message Identity element. This is a string element (or attribute) that precedes the embedded message in the model and whose Interpret Value As property is set to Message Identity.

When a multipart message is input to a message flow, the Message Identity element should have a value which corresponds to either the Name or the Message Alias of the next embedded message in the bitstream. This enables the MRM parser to correctly identify the embedded message in the model.

For cases where the Message Identity element value does not match the Name of the message, you should use the Message Alias property to specify this value. The MRM parser will try to match on Name first and if that fails it will try to match on Message Alias.

Once the MRM parser has encountered a Message Identity element, its value applies to all embedded messages that are contained immediately within the current message. This does not apply to embedded messages within embedded messages; any embedded message must have its identity provided by a Message Identity element within its immediate parent message.

If a second Message Identity element is encountered within the current message, its value overrides any previously held. This enables different peer embedded messages to exist within a given message.

Message Identity takes priority over Message Path. If both are specified then Message Identity is used. You are recommended to use just one of these techniques for a given multipart message.

## Embedded messages defined in different message sets

By default an embedded message is assumed to be defined within the same message set as the current message. This can be overridden using a Message Set Identity, which works in a very similar manner to a Message Identity.

An embedded message defined in a different message set must have its message set identified earlier in the message using a Message Set Identity element. This is a

string element (or attribute) that precedes the embedded message in the model and whose Interpret Value As property is set to Message Set Identity.

When a multipart message is input to a message flow, the Message Set Identity element should have a value which corresponds to either the Identifier, Name or Message Set Alias of the message set defining the next embedded message in the bitstream. This enables the MRM parser to correctly identify the message set to use.

For cases where the Message Set Identity element value does not match the Identifier or Name of the message set, you should use the Message Set Alias property to specify this value. The MRM parser will try to match on Identifier first, then on Name, and finally Message Set Alias.

Once the MRM parser has encountered a Message Set Identity element, its value applies to all embedded messages that are contained within the current message. It also applies to embedded messages within embedded messages, unless an embedded message also contains a Message Set Identity element.

If a second Message Set Identity element is encountered within the current message, its value overrides any previously held. This enables peer embedded messages to reside within different message sets.

The following example of an X12 message shows the use of both Message Identity and Message Set Identity. The field containing 004010X092 within the GS segment on line 0002 holds the Message Set Identity as a Message Set Alias. The 207 on line 0003 in the ST segment is the Message Identity held as a Message Alias. The embedded message is from line 0004 to 0015 inclusive.

**Note:** The line numbers and spaces at the beginning of each line are for illustrative purposes only and do not exist in the actual message.

```
0001  ISA*00*          *00*          *30*12-3456789      *ZZ
      *9876543-21      *000104*1820*U*00401*000000001*0*T*:*!
0002  GS*HS*HOSP CLAIM*PAYER ADJDEPT*20000104*1820*1*X*004010X092!
0003  ST*270*1234!
0004  BHT*0022*13*10001234*19990501*1319!
0005  HL*1**20*1!
0006  NM1*PR*2*ABCCOMPANY*****PI*842610001!
0007  HL*2*1*21*1!
0008  NM1*1P*2*BONE AND JOINT CINIC*****SV*2000035!REF*N7*234899!
0009  N3*55*HIGH STREET!
0010  N4*SEATTLE*WA*98123!
0011  HL*3*2*22*0!TRN*1*93175-12547*9877281234!
0012  NM1*IL*1*SMITH*ROBERT*B***MI*11122333301!
0013  REF*1L*599119!
0014  DMG*D8*19430519*M!
0015  DTP*472*RD8*19990501-19990515!EQ*30**FAM!SE*17*1234!
0016  GE*1*1!IEA*1*000000001!
```

## Physical format considerations

Both Message Identity and Message Set Identity are applicable to all physical formats. Versions of the TDS physical format prior to Version 6.0 included embedded message identification by Message Key which worked in a similar manner to Message Identity. Message Key has been deprecated and is superseded by Message Identity.

*Identifying an embedded message using a Message Path:* The Message Path technique for identifying embedded messages is useful when the multipart message contains no information as to the identity of an embedded message.

In figure 1 below, the Message Header and Message Trailer act as an envelope for the message body. Typically they have a fixed structure, but the Message Body can be defined with many different structures.

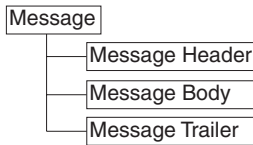


Figure 2. Multipart message structure

A place holder for an embedded message is created by setting the Composition property of the complex type or group of the Message Body element to Message. This allows an embedded message to be added at this point within the outer Message, thus creating a multipart message.

When using the Message Path technique to parse such a multipart message, the embedded message must be identified by a fixed path to the innermost message from the outermost message. For this example this would simply be:

Message/Message Body

If the path to the innermost message contains intermediate elements then they must be included in the path as well. In the following example these elements are shown in bold:

Message/**Data1/Data12**/Message Body

This technique can be used to identify nested embedded messages as well, by simply extending the path. For example:

Message/**Data1/Data12**/Message Body/**Data2**/Inner Message

The path is specified using one or both of two properties, the Message Type property of a WebSphere Message Broker input node (or MQRFH2 header) and the Message Type Prefix property of the containing message set. These two properties are combined together to produce a final path that is used to locate embedded messages.

Message Identity takes priority over Message Path. If both are specified then Message Identity is used. You are recommended to use just one of these techniques for a given multipart message.

**Note:** It is not possible to use the Message Path technique to identify multiple peer embedded messages.

## Embedded messages defined in different message sets

This is not supported by the Message Path technique.

## Physical format considerations

Message Path is applicable to all physical formats.

### Cardinality: optional, repeating and mandatory elements:

*Elements:* The number of occurrences of an element can be controlled using the properties Min Occurs and Max Occurs. Using these properties, an element can be defined as mandatory, optional or repeating.

- A mandatory element has Min Occurs  $\geq 1$ . A mandatory element must occur at least once in an input message.
- An optional element has Min Occurs = 0. An optional element can be omitted from the input message.
- A repeating element has Max Occurs  $> 1$  or a Max Occurs of -1 to indicate unlimited repeats are allowed. It can occur more than once in the input message, and all the occurrences must appear together without any other elements between them.

If a complex type or a group contains two (or more) members which refer to the same global element, the second reference is a duplicate. This is different from a repeating element, because the two references will usually be separated by other members of the type or group. In the input message, the second occurrence will usually not appear immediately after the first occurrence. Duplicate element references are not allowed within types and groups which have compositions of Choice, OrderedSet or UnorderedSet.

*Attributes:* The number of occurrences of an attribute can be controlled by setting it to required, optional or prohibited.

- A required attribute is similar to a mandatory element - it must occur in the input message.
- An optional attribute is similar to an optional element - it can be omitted from the input message.
- A prohibited attribute must not appear in the input message.

An attribute is not allowed to repeat, and duplicate attribute references are not allowed within an attribute group.

**Predefined and self-defining elements and messages:** An instance element is predefined if it is possible to find a matching element definition in the message model with an appropriate set of properties and in the correct context. Otherwise, it is self-defining. Similarly, an entire message is self-defining if no corresponding message is present in the message model.

This topic discusses the processing of elements which appear in an instance message, but which cannot be matched with any element in the message model.

Self-defining elements can only be used when the physical format of the message is a tagged format such as XML or TDS. If your physical format is fixed-length (C or COBOL records) or delimited (e.g. comma separated) then you must ensure that your message model defines every message and every element which needs to be parsed.

If you have chosen not to model your messages, or if no message sets have been deployed to the broker, all messages and elements will inevitably be self-defining. In this situation, it is not possible to influence the parsing and writing of elements



via the message definitions; the self-defining elements will be parsed and written according to the default behaviour of the parser and writer.

Self-defining elements, and all elements within a self-defining message, are not validated against value constraints, and any missing fields are not assigned default or fixed values.

However, if an element can be matched against the message model, the parsing and the writing of the element is guided by the logical and physical formats which have been defined. This provides a range of benefits, all of which arise from the information provided to the broker through the message model.

**Substitution groups:** Substitution groups are an XML Schema feature which provides a means of substituting one element for another in an XML message. A substitution group is a list of global elements which can appear in place of another global element, called the head element.

A substitution group is defined by setting the substitution group property on one global element (the member element) to point at another global element (the head element). This adds the member element to the substitution group of the head element.

**Note:** If you are generating dictionaries from your message set, and deploying them to WebSphere Message Broker, you should read the “Schema restrictions” on page 15 topic, and note the section which relates to substitution groups.

**Tip:** If your messages are never rendered as XML, or if you have a simple message model, you should consider using a complex type or a group with Composition set to Choice, instead of using substitution groups.

## Elements

### Head elements

A head element is simply an element which can be substituted. When a message is parsed, one of its member elements can appear in place of the head element without causing a validation error.

### Abstract elements

An abstract element is a head element which must be substituted. The ‘abstract’ attribute on the element indicates this. Usually, abstract elements will have other elements in their substitution group - otherwise they are of little use. Wherever an abstract element appears in a message definition, a member of its substitution group must appear instead.

## Attributes

### The block attribute on elements

The block attribute on an element limits the set of global elements which can substitute for the element. The block attribute can take any subset of the values restriction, extension, substitution, or all.

- If the block attribute contains restriction, an element which is based on a restriction of the element’s type cannot substitute for the element.
- If the block attribute contains extension, an element which is based on an extension of the element’s type cannot substitute for the element.

- If the block attribute contains substitution, an element which is a member of the element's substitution group cannot substitute for the element.
- If the block attribute contains all, all of the above limits apply.

#### **The final attribute on elements**

The final attribute on an element limits the set of global elements which can be a member of the element's substitution group. The final attribute can take any subset of the values restriction, extension or all.

- If the final attribute contains restriction, an element which is based on a restriction of the element's type cannot be in the substitution group of the element.
- If the final attribute contains extension, an element which is based on an extension of the element's type cannot be in the substitution group of the element.
- If the final attribute contains all, both of the above limits apply.

#### **The block attribute on complex types**

The block attribute on a complex type limits the set of other types which can substitute for that type. The block attribute can take values restriction, extension, or all. The meanings for these values are the same as those shown for the block attribute on an element above. An element which is a member of a substitution group can only substitute for the head element if its type is compatible with the block attribute on the type of the head element.

#### **Default block and final attributes**

A default for the block and final attributes can be set at the message definition file level. If a default for one or both of these attributes has been set and the relevant block or final attribute has not been set at the object level, the default setting will be used for that object. You can override the default setting at the object level.

### **Message categories**

Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set. A message category provides another way of grouping your messages, perhaps for documentation or convenience purposes, or for assisting in the generation of Web Services Description Language (WSDL) files.

A message set category file is created using the New Message Category File wizard.

When you have created your message category file, you must specify one key property.

#### **Message Category Kind**

This indicates whether the message category is to participate in the generation of WSDL files. See "WSDL" on page 77.

You can then add messages to the message category file. If the message category is to participate in WSDL generation, you should assign appropriate values to the Role Name and Role Type properties of each member message.

**Message category identification:** The name of a message category is provided by the name of the `.category` file. You can change the message category name by renaming the file.

## Namespaces

XML instance documents and XML Schemas can make use of namespaces. Namespaces provide a method to qualify object names.

A single XML instance document can contain elements and attributes that are defined for and possibly used by multiple applications. Two different elements or attributes within the same document might require the same name. Individual applications need to be able to recognize the elements and attributes which they are designed to process. In circumstances such as this, the definitions can be distinguished from each other by qualifying each element with a different namespace. This avoids problems of name collision and mistaken recognition.

XML Schemas can define a target namespace. Global elements, attributes, groups and types defined within an XML Schema are qualified by the target namespace if it has been defined. Optionally local elements and attributes can also be qualified by the target namespace. Thus namespaces assist in the development of a library of XML Schemas which can be developed independently. Providing the namespace name used for an XML Schema is unique a developer does not have to be concerned about name clashes with objects defined within other XML Schemas.

The scope of a namespace extends beyond that of its containing document and is identified by a Uniform Resource Identifier (URI). In order to serve its purpose, a URI should be unique. You might be more familiar with the concept of a Universal Resource Locator (URL). URIs often use the same syntax as URLs, although the URI definition is broader than the specification of a URL. This is an example of a URI: `http://mycompany.com/xml_schema`

A namespace prefix is declared as a shorthand for the full URI name and this is used to qualify all elements belonging to that namespace. The prefix to be substituted for a namespace in an XML instance document or XML Schema is specified using an `xmlns` or `xmlns:c` attribute. A default namespace can also be defined using an `xmlns` or `xmlns:c` attribute. If a default namespace is defined any element or attribute with no prefix is qualified with the default namespace. If no default namespace is defined any element or attribute with no prefix is unqualified by a namespace.

### The message model

The message model provides the ability to support namespaces within message sets. However you can choose whether you wish to enable or disable namespaces for your message set. If you choose to disable namespaces when you create your message set you can enable namespaces at a some later point. However once you have enabled namespaces for a message set you cannot disable namespaces.

A single message set which has namespaces enabled can contain a number of different namespaces. Each namespace is represented by a different Message Definition File. When you create a Message Definition File you can choose whether it is to have an associated namespace or whether it is be in the notarget namespace. If you choose to associate a namespace with a Message Definition File you must also choose a prefix.

If the Message Definition File has an associated namespace the following global objects are qualified with the namespace:

- Elements
- Attributes
- Simple Types

- Complex Types
- Groups
- Attribute Groups

Optionally, local elements and attributes can be qualified with the namespace.

Objects defined within a Message Definition File can reference objects in other Message Definition Files within the same message set. This is achieved by importing or including one Message Definition file within another.

### **The XML Wire Format**

The namespace associated with a message definition file is part of the logical layer of the message model. Therefore it is not dependent on an XML Wire Format being present. However if you have an XML Wire Format the namespace information from the logical layer is used to populate some of the properties of the XML Wire Format. If namespaces are enabled for a Message Set, in the XML Wire Format a table of namespace URI/prefix pairs is maintained. This table is initially populated with the namespaces of all of the Message Definition Files with their prefixes when they were created.

If you are using WebSphere Message Broker and your message set has namespaces enabled, the Broker does not store the values of any `xmlns` or `xmlns:c` attributes in the tree when it parses an XML instance document. It also does not store the values of any Schema Location and No Namespace Schema Location attributes. This is because when an XML document is written out the Broker regenerates this information from the properties of the XML Wire Format of the message set.

If you are using WebSphere Message Broker the table of Namespace URI/prefix pairs is used by the MRM Domain when outputting an XML message. Elements and attributes that are qualified by a namespace are prefixed with the corresponding prefix from the table. The Broker also manages the output of the corresponding `xmlns` or `xmlns:c` attributes which map the prefixes to namespaces. You can choose whether `xmlns` or `xmlns:c` attributes for all of the entries in the Namespace URI/prefix table are output at the start of the document or whether they are only output in the document when required.

If namespaces are enabled for a Message Set, in the XML Wire Format there is a table of schema locations that map namespace URIs to file names. You can add entries to this table and you can also map a file name to the notarget namespace. If you are using WebSphere Message Broker this table is used to output `schemaLocation` and `No Namespace Schema Location` attributes at the start of the XML document.

### **Message parsing and ESQL**

If you are using WebSphere Message Broker the MRM domain and XMLNS and XMLNSC domain parsers recognize prefixed names in the XML messages they parse and internally map these to the correct namespace. Elements and attributes in the dictionary generated from the message model can either be qualified or unqualified with a namespace as discussed in the message model section.

If you are using XML format in the MRM domain, then elements or attributes are matched based on the namespace in the dictionary when the parsed message is matched against the dictionary generated from the

message model. Thus for an element or attribute in a message to match with the dictionary both its name and namespace must match.

If you are using WebSphere Message Broker, support is provided to allow you to specify namespaces when writing ESQL. It is not necessary to write ESQL which is namespace aware if you are not using namespaces. However, if you decide to use namespaces, your message definition files can target any namespace which you choose and it will be necessary to write namespace aware ESQL. The namespace in which an element resides is stored in the message tree when parsed. This is a logical property and is held regardless of the physical wire format in which messages are parsed and written. New syntax has been added to ESQL to make it easy to reference elements namespaces using defined prefixes.

### **Importing from other formats**

The message model allows you to create Message Definition files from other formats by importing them into the Message Brokers Toolkit.

- If you import an XML DTD file the Message Definition File created will be in the notarget namespace.
- If you import an XML Schema file, the target namespace of the created Message Definition File depends on whether namespaces have been enabled for the message set.
  - If namespaces are enabled then the target namespace of the Message Definition File created will be the target namespace of the XML Schema being imported.
  - If namespaces are disabled for the message set then the created Message Definition File will be in the notarget namespace. This type of import does not provide full namespace support. If you are using WebSphere Message Broker, you will not have to write namespace aware ESQL or Java to process an XML message parsed against the dictionary generated from this message model. For reasons why you might want to do this, see “Importing XML Schema into message sets with namespaces disabled” on page 71
- If you import a COBOL Copybook or a C Header file, the target namespace of the created Message Definition File depends on whether namespaces have been enabled for the message set.
  - If namespaces are enabled then the target namespace of the Message Definition File created is the notarget namespace. This default namespace can be overridden by specifying a target namespace in the New Message Definition File wizard. For reasons why you might want to do this, see “Namespaces with non-XML messages.”
  - If namespaces are disabled for the message set then the created Message Definition File will be in the notarget namespace

**Further information about XML:** On the World Wide Web Consortium (W3C) Web site, also see:

- Extensible Markup Language (XML)
- XML Schema Part 0: Primer
- Namespaces in XML

**Namespaces with non-XML messages:** Use of namespaces by WebSphere Message Broker is not necessarily limited to XML message models. There is one scenario where use of namespaces by non-XML message models can simplify the ESQL or Java code that you write.

Before describing this scenario, it is important to understand that the MRM parser, when parsing messages that are defined in a Message Definition File that has a target namespace, produces a logical message tree containing both name and namespace information. It does this *regardless of the physical format of the message*. For non-XML (CWF or TDS) messages, the namespace is obtained from the Message Definition file.

Consider a transformation scenario where a message from a legacy COBOL application requires transforming into namespace-aware XML, for example a SOAP XML message. The transform must map the logical message tree created for the COBOL message to a logical message tree that matches the XML message. If the COBOL message tree does not contain namespace information then each mapping from a COBOL field to an XML element must set the namespace for the XML element. However, if the COBOL message tree already contains the required namespace information then this mapping is much simpler.

To enable the MRM parser to create namespace information in a message tree created from a CWF or TDS message, you need to specify a target namespace for the Message Definition File. This must be done as part of the Message Definition File creation process; you can not do this once the file has been created. There are two ways to specify a target namespace. For each of these, make the target namespace of the Message Definition File the same as the target namespace of the XML message into which the non-XML message is being transformed.

- If you are creating your non-XML message model by hand in the message editor, use the New Message Definition File wizard to specify a target namespace.
- If you are importing from COBOL or C, use the New Message Definition File wizard or the `mqsicreatemsgdefs` command options file to specify a target namespace.

When dealing with both the message tree for the non-XML message and the message tree for the XML message, the ESQL or Java code that you write to perform the transformation must be namespace aware.

**Reusing message definition files:** One Message Definition File can reuse message model objects defined in another Message Definition File. There are two mechanisms that XML Schema provides to do this: `import` and `include`. The namespaces of the two files determine which of `import` or `include` should be used:

	Target file has a target namespace	Target file has notarget namespace
Parent file has a target namespace	<code>xsd:import</code>	<code>xsd:include</code> <sup>1</sup>
Parent file has notarget namespace	<code>xsd:import</code>	<code>xsd:include</code>

1. When a target namespace file includes a notarget namespace file, referencing an object in the target file from the parent file causes the object to appear in the namespace of the parent file.

When `import` or `include` are used, global objects from the target file can be used in the parent file. For example, an element in the parent file can be given a complex type defined in the target file.

The namespace of objects in the target file is preserved in the parent file, with the exception noted in the previous table of a target namespace file including a notarget namespace file. This exception is sometimes called the chameleon namespace effect.

Chameleon namespaces have limited support when used with the MRM domain. When referenced in the parent file, the objects in the target file appear in the namespace of the parent file, but they are assigned default physical format information. In other words, physical format information defined in the target file is not available for use in the parent file. Only use Chameleon Namespaces in the MRM domain to model XML messages where physical format information has not changed from the default.

XML Schema provides a variation of `xsd:include` called `xsd:redefine`, which is not supported by WebSphere Message Broker. Using `xsd:redefine` will give a task list error. A Quick Fix is offered to convert occurrences of `xsd:redefine` into `xsd:include`.

### **Model integrity**

When you create your model, it is important that it is internally consistent and is capable of being generated into the form you desire, such as a WebSphere Message Broker message dictionary or an XML Schema document. To assist with this, each time you save a message set file it is validated as follows:

#### **Logical validation**

This validation ensures that the logical model is correct. For message definition files, this involves ensuring that the rules of XML Schema have been correctly followed.

#### **Physical validation**

This validation ensures that any physical formats you have specified for your model have been correctly populated. There are a set of checks for each of the MRM domain physical formats CWF, XML and TDS. This ensures that the MRM parser will be able to parse and write messages that conform to your model.

Once validation has taken place, any errors or warnings are shown in the task list. Double clicking on a task list entry will open the file and position the editor at the object in error. You are recommended to organize the task list so that errors are shown before warnings. In that way, errors are not hidden. The task list provides a comprehensive filtering capability if you wish to hide low priority warnings or warnings that you are know about and are comfortable with.

The generation of a message dictionary or an XML Schema is prevented if any errors are present. The presence of warnings alone does not prevent generation, but high priority warnings must be reviewed as a model generating such warnings might be incomplete.

Where task list warnings or errors occur, these are listed in the Problems view of the Broker Application Development perspective. While a majority of these will require you to manually investigate and resolve them, a number of warnings and errors that meet specific criteria can be repaired using a quick fix process.

## **Physical formats in the MRM domain**

Each message definition file describes both the logical structure of your messages, and the physical formats that describe the precise appearance of your message bit

stream during transmission. If you are using the MRM domain, physical format information must be provided, as it tells the parser exactly how to parse the message bit stream.

You can think of a message as a packet of data that is sent from one place to another. The sender and receiver of the message will have agreed the structure of the message and what each field in the message means. This is the platform and protocol independent logical structure.

The sender and receiver will have also agreed on the physical representation of the message, how the data is physically laid out on the wire. For example, if you define a message that conveys information about a debit of an individual's bank account, it can be represented in different physical forms (examples are XML, or a fixed structure such as a COBOL copybook). The meaning and data is the same in both cases: only the physical layout has changed.

If you are using the MRM domain, you can model a variety of different physical representations using named physical formats.

- Use the Custom Wire Format (CWF) physical format to model fixed format messages from legacy applications written in C, COBOL, PL/1 and other languages. This support includes the ability to create a message model directly from a C header file or COBOL copybook.
- Use the XML Wire Format (XML) physical format to model XML messages, including those that exploit XML namespaces. This support includes the ability to create a message model directly from an XML DTD or XML Schema file.
- Use the Tagged Delimited String Format (TDS) physical format to model formatted text messages, perhaps with field content identified by tags or separated by specific delimiters or both. This support is rich enough to model industry standards such as SWIFT, EDIFACT and X12.

## Different physical representations

The following example shows how a very simple logical message can have different physical representations.

The logical model defines the structure and order of the message. In the following example, the three fields are simple integers, and C follows B, which follows A:

```
int A;  
int B;  
int C;
```

- A typical Custom Wire Format representation for this logical message would be 12 bytes of data, with each of A, B and C occupying 4 bytes. Alternatively, perhaps A is 4 bytes long, but B and C are only 2 bytes long. You supply the precise physical information for each field in the message as CWF properties.
- A typical XML representation of this model is as follows:

```
<Msg><A>xxxxxxx</A><B>xxxxxxx</B><C>xxxxxxx</C></Msg>
```

where xxxxxxx is the value of the integer represented as a string (XML deals only with strings). An alternative representation might be:

```
<Msg A="xxxxxxx" B="xxxxxxx" C="xxxxxxx"/>
```

where the values of the integers are stored as XML attributes rather than XML elements. You supply the precise XML rendering for each field in the message as XML properties.



- TDS allows several different representations to be modeled. Each integer could be preceded by a tag to identify it and a delimiter to terminate it, as follows:  
`{A_tag:xxxxxxx;B_tag:xxxxxxx;C_tag:xxxxxxx}`

An alternative might rely on the data being ordered so only the terminating delimiter needs to be specified, as follows:

`[xxxxxxx;xxxxxxx;xxxxxxx]`

You supply the precise identification regime as TDS properties.

This shows that the logical model is unchanged. It is constant, regardless of the physical representation that you choose to model on top of it, using the physical format support provided by the MRM domain. The MRM parser is able to transform the message from the input physical representation to any number of output representations, based on the wire format layers that you have defined.

### Creating physical formats

Once you have created your message set, you can create physical formats. You do this using the Message Set Editor. When you next save the messageSet.mset file, any new physical formats are added to all the objects in all the message definition files in that message set.

The next time you edit an object in a message definition file, you will see the physical formats in the properties hierarchy pane of the Properties tab. If you click on a physical format for an object, you will be presented with a property sheet where you can enter the information for that physical format for that object.

Note that not all objects have properties in all physical formats. For example:

- CWF properties only apply to local elements and attributes, and element and attribute references.
- Complex types and groups only have TDS properties.
- Messages only have XML properties.

This is due to the different nature of each physical format, and these differences are explained in more detail in the section for each physical format.

There is no limit to the number of physical formats you can create in a given message set. However there are some recommendations that apply if you want to mix physical formats of different kinds in the same message set.

Physical formats can be deleted if no longer required.

### Custom wire format

Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed format data structures or elements, which are not separated by delimiters.

Within a CWF messaging environment, it is not possible to distinguish one element from the next without knowledge of the message structure. To correctly determine the values of individual elements, the following information must be made available to the message parser:

- The order (this is defined in the logical properties)
- The length (can be specified in bytes, characters or character units)
- The cardinality (that is, the number of repeats)

- The type of data contained in each element (this is partly defined in the logical properties but can be further qualified in the CWF physical format)
- A number of characteristics based upon the logical type of the data

A CWF physical format is typically used to describe messages which are mapped to a C structure, a COBOL copybook or other programming language data structure definition.

You can add more than one CWF physical format to a message set, but within that message set, each physical format must have a unique name. When parsing a CWF message using the MRM parser, the wire format name specifies the physical properties that are to be used by the parser.

Adding a CWF physical format to a message set allows you to process input messages and construct output messages in this format. Messages can be transformed between CWF and the other physical representations (for example TDS or XML). Note that while the other physical representations support self-defining elements (that is elements which do not have a definition in the logical model) within the MRM domain, the parsing of a CWF message does not. Consequently, any such self-defining elements are discarded during the output of messages in CWF format.

**Custom wire format - Model integrity:** When you save a message definition file, the definitions that it contains are checked to ensure that they make sense and provide sufficient information about the message. This action is called validation.

The CWF physical format depends on fixed format data structures, so the majority of tests applied to a CWF message confirm that each fragment of a message (and therefore the message as a whole) has a well defined length. Thus, these tests examine properties such as Length Count, Length Reference and Length Units.

Typically, one or other of Length Count and Length Reference must be set. If Length Reference is set, then it must refer to an element that is of simple type integer and that appears earlier in the message than the current item.

Tests other than these tend to be both simple and obvious so that, for example, the message set property First Day of Week has to be the name of a day in the week.

The fact that CWF relies on fixed format data structures also imposes some limitations on the messages that can be represented:

- CWF cannot represent a message that includes the use of schema wildcards. (This is a consequence of its inability to handle undefined content).
- CWF cannot represent a message that includes recursive definitions.
- CWF cannot represent a message that includes the use of substitution groups, since there is no way to recognize the substituted element.

**Custom wire format - NULL handling:** CWF supports the handling of explicit NULL values within messages. An explicit null is identified by a specific value that identifies an element as being null.

The Boolean Null Value can be specified at the message set level and applies to the boolean elements of all messages defined in that message set. The null value for all other element types can be specified individually for each element.

CWF supports the representation of null values using the Encoding Null and Encoding Null Value element properties. Together this information controls how null values are handled by the MRM parser.

The Encoding Null property can be set to one of four values:

#### **NullLogicalValue**

The Encoding Null Value property is interpreted as a logical value. Thus, if its value is set to 0 for example then both 0 and 0.00 are interpreted as null values.

#### **NullLiteralValue**

The Encoding Null Value property is interpreted as a string value. Thus, the value of the element in the bitstream must match exactly the value specified to be interpreted as a null value.

#### **NullPadFill**

This should be used for fixed length elements. On output, any element with a null value will be padded to the appropriate length with the specified padding character.

#### **NullLiteralFill**

The Encoding Null Value property is interpreted as a single character string value. Therefore, each character of the value of the element in the bitstream must match exactly the character value specified, to be interpreted as a null value.

**Custom wire format - Multipart messages:** CWF can only use the Message Type property to resolve the hierarchy of the first embedded message structure. If you have defined the outer message to contain more than one complex type that defines embedded messages, you must resolve the message choice within the second and subsequent complex types using ESQL.

The Custom Wire Format (CWF) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

Alternatively an embedded message may be resolved by using ESQL to identify the message. The first message that you reference in this way is assumed to be the selected message. This works in an identical manner to unresolved choice handling.

**Custom wire format - Data Conversion:** Custom Wire Format supports the conversion of data to a different code page (for schema string simple types) or encoding (for schema numeric simple types), or both.

A message set contains properties to enable the character (CCSID) and numeric encoding (Byte Order / Float Format) information to be specified. If you generate a message dictionary for deployment to a WebSphere Message Broker, then this information can be overridden using the appropriate fields of the WebSphere MQ message header.

**Custom wire format: relationship to the logical model:** There are some restrictions for messages defined using a CWF in relation to the logical model and these are discussed below.

*Composition:* A CWF message is always output with the elements in the sequence specified in the logical message model definition. However, you do not always have to specify ESQL that builds the elements in that sequence. The following rules for coding ESQL are given for each value for the type Composition property.

### **Sequence**

You must build the output message to match the sequence of the elements/groups in the message. You can normally do this using ESQL SET statements to assign a value to each element or type. The first SET statement sets the value of the first element or type in the message, the second SET sets the value for the second element or type, and so on. You can vary this sequence of statements using ESQL ATTACH, CREATE, and MOVE statements.

If the elements or types have default values, and you do not build the message in the correct sequence, those elements built out of sequence will contain their default values, not the values that you set. This is because elements built out of sequence are assumed to be self-defining, and for CWF these are discarded when the message is written to the bit stream.

### **Ordered Set**

You must build the output message to match the sequence of the elements in the message. You can normally do this using ESQL SET statements to assign a value to each element. The first SET statement sets the value of the first element in the message, the next SET sets the value for the second element, and so on. You can vary this sequence of statements using ESQL ATTACH, CREATE, and MOVE statements.

If the elements have default values, and you do not build the message in the correct sequence, those elements built out of sequence will contain their default values, not the values that you set. This is because elements built out of sequence are assumed to be self-defining, and for CWF these are discarded when the message is written to the bit stream.

### **Unordered Set**

You can build elements of the output message in any sequence. On output, the elements will be written in the order specified in the logical message model definition.

**All** You can build elements of the output message in any sequence. Each element must only be present once (that is, it must not repeat). On output, the elements will be written in the order specified in the logical message model definition.

### **Choice**

A choice cannot be resolved purely from the data. The receiving program must interpret the data and decide which option of the choice the message instance contains. This process is known as unresolved choice handling. The first reference in the application to any one of the choice elements resolves the choice to the option containing that element.

### **Message**

In CWF, there is no way to identify, within the bitstream of an input message, which embedded message(s) appear within this input message. Mechanisms for the resolution of embedded messages is discussed in the "Custom wire format - Multipart messages" on page 43 topic.

*Content validation:* CWF is a fixed format and all elements must be present in a message. Therefore, [Content validation] is ignored. On output, all elements must

be set explicitly (for example, using ESQLE SET), set implicitly (using a tree copy function), or must have a default value defined.

*Default values:* On output of a CWF message in the MRM domain, any element, or occurrence of an element for which a value has not been set (either explicitly or implicitly) inherit the element's specified default value. If no default value has been specified then an exception is thrown.

*Min Occurs and Max Occurs:* The logical properties Min Occurs and Max Occurs specify the permitted number of occurrences of an element or group in a message. They are used when parsing and writing messages, and when validating the content of a message.

In CWF, Max Occurs occurrences are expected when parsing, and Max Occurs occurrences are output when writing with default values used for missing elements.

- A varying number of occurrences (Min Occurs <> Max Occurs) is ignored, Max Occurs is assumed.
- Optional occurrence (Min Occurs = 0) is ignored, Max Occurs is assumed.
- Always absent (Max Occurs = 0) is allowed.
- An unbounded number of occurrences (Max Occurs = -1) is allowed if the element or group is the last child in its parent group, and the group is terminated by the end of the message bit stream. On writing, the writer outputs all occurrences in the message tree, if this number is less than Min Occurs then additional default values are written.

However this behavior is overridden if the Repeat Reference property is set which indicates that the number of occurrences is given instead by an integer element that occurs earlier in the message. In this case Max Occurs is ignored.

When validating, Min Occurs and Max Occurs are both used to check that the content of the message tree matches the model.

*Simple types – lists and unions:* Lists and unions are XML-specific concepts. An element or attribute of a simple type that is a list or a union will cause a task list warning if a CWF physical format is present in the message set. The user can choose whether to make this an error, warning, or information by editing the Validation preferences. The dictionary generator will omit messages defined to contain such elements or attributes from the CWF section of the dictionary.

## **XML wire format**

The XML wire format is the physical representation of a message that can be parsed as XML.

An XML wire format describes the physical representation of a message that is written according to the standards given in the W3C Extensible Markup Language (XML) specification. The wire format defines information that is used to parse or write XML messages in a runtime environment such as a broker. XML versions 1.0 and 1.1 are both supported.

You can add more than one XML physical format to a message set, but within that message set, each physical format must have a unique name. The default name for an XML wire format is XML1. Typically, you might do this if a single logical message can be represented with its components rendered in different ways, for

example as the value of an element in one case but as the value of an attribute in another. The physical format name identifies the definitions that are to be used by the message broker at runtime.

After adding an XML physical format, all XML properties for all existing objects in the message set will be set to default values. Thus, immediately after adding the format and deploying the message set to a runtime environment, it is possible to process XML messages using MRM features.

You can configure XML properties for the message set, and for objects within the message set. Objects that can have XML properties are messages, elements, and attributes. For example, a message object can be customized to define a specific DTD declaration on output; an element can have a tag name assigned to it which is different from its MRM element name.

Adding an XML wire format to a message set allows you to both process input messages, and to construct output messages in this format. You can also transform messages between XML and either CWF or TDS.

XML messages are, by their nature, self-describing: each piece of data is prefixed by a tag name or an attribute name. Therefore, it is possible for an XML message instance to contain elements which are not in the MRM definition for that message.

- If such an element exists in the message set, the MRM objects for that element are used in parsing or writing the message.
- If the element does not exist in the message set, it is treated as a self-defining element and its data type is set to string.

Although it is possible to define an XML message using the Message Definition Editor, WebSphere Message Broker also provides importers for both XML Schema and DTD, and these are often quicker and easier than manual definition.

**XML wire format - Model integrity:** When you save a message definition file, the definitions that it contains are checked to ensure that they make sense and provide sufficient information about the message. This action is called validation.

For XML, these checks mostly concern the uniqueness and validity of XML names in global elements and attributes, and also for elements and attributes within complex types or groups. Since elements can be rendered as attributes, and attributes as elements, uniqueness tests will sometimes detect conflicts between objects that are not obviously related, such as an attribute and an element.

Tests other than these tend to be both simple and obvious so that, for example, the message set property First Day of Week has to be the name of a day in the week.

**XML wire format - NULL handling:** The purpose of null handling is to specify how messages will deal with null values, that is, the absence of a meaningful value for an element. Null properties for XML are set for the message set only, and apply to all the defined objects within the message set, using the four properties Encoding Null Num, Encoding Null Non-Num, Encoding Null Num Val and Encoding Null Non-Num Val.

The purpose of these parameters is to specify how messages deal with null values. In an XML message there are several options. Most obviously an element could simply omit a value, for example:

```
<element1></element1>
```

Alternatively, the element could include a distinctive value that means that no real value is present, for example.

```
<element1>null</element1>
```

The properties Encoding Null Num and Encoding Null Non-Num specify the style of null handling, for example, that null is represented by an empty element.

The properties Encoding Null Num Val and Encoding Null Non-Num Val provide a value (if needed) to represent a null value. For an element of type string, this might be null or unspecified while for a number it might be 0 or 0.0.

**XML wire format - Multipart messages:** The XML Wire Format supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message. It is not mandatory for them to be used in order to identify an embedded message, as the content and the structure of the message are both fully described by the bitstream. The logical model must include an embedded message definition at the appropriate position, but there are no extra definitions required.

In the MRM domain, embedded XML messages are recognized by matching their XML tag name against the XML name of a message definition in the message model. If the position of the embedded message corresponds to an embedded message definition in the logical model, the message is recognized.

So when either of Message Identity or Message Path techniques are used with XML messages, the Message Identity element value or the Message Path value is only used to check that the message identified by the XML tag in the bitstream is the one that is expected. If it is not the expected message then an exception is thrown.

Where you have defined the embedded message in a different message set, you will need to use a Message Set Identity element value to specify the target message set.

If the embedded message definition is a complex type, the message definition will contain a complex element based on that complex type. This complex element will have its own tag, which will appear in the bitstream before the tag for the embedded message. If you want to avoid this extra tag, you can create the embedded message definition from a group, and insert the group at the appropriate position in the message model.

**Tip:** Note that the root tag property of an embedded message is not applicable.

**XML wire format - Data conversion:** All XML message data is handled as strings. All data is therefore subject to CCSID conversion only.

**XML wire format: relationship to the logical model:** The XML wire format generally respects all of the setting in the logical model, but shares certain restrictions in common with the other physical formats. These restrictions are documented in “MRM model restrictions” on page 674.

## Default values

The XML wire format ignores default and fixed values on elements and attributes. If validation is enabled in WebSphere Message Broker, this can lead to unexpected

validation errors for missing elements, even though they have default or fixed values.

## Simple types – unions and lists

The XML properties of an element or attribute of a simple type that is a union or list vary depending on the members of the union or the itemType of the list. If the union or list includes a dateTime type (or other date/time related type) the Date Format field will be displayed. If the union includes a binary type, the Encoding field will be displayed.

## Min Occurs and Max Occurs

The logical properties Min Occurs and Max Occurs specify the permitted number of occurrences of an element or group in a message. They are used when parsing and writing messages, and when validating the content of a message.

When parsing and writing, Min Occurs and Max Occurs are effectively ignored. When parsing, the number of occurrences is identified by the tags in the message. When writing, the writer outputs all occurrences in the message tree.

- A varying number of occurrences (Min Occurs <> Max Occurs) is allowed.
- Optional occurrence (Min Occurs = 0) is allowed.
- Always absent (Max Occurs = 0) is allowed
- An unbounded number of occurrences (Max Occurs = -1) is allowed.

When validating, Min Occurs and Max Occurs are both used to check that the content of the message tree matches the model.

**Handling xsi:type attributes using the XML physical format:** The prefix "xsi" is the namespace prefix used by convention for the XML Schema namespace for instances. XML documents can contain elements that have an xsi:type attribute. This provides an explicit data type for the element.

The MRM XML parser is sensitive to xsi:type attributes in the XML document. It modifies the data type of the element accordingly and adds the xsi:type attribute into the message tree.

The MRM XML writer is sensitive to xsi:type attributes in the message tree. It outputs xsi:type attributes according to XML Wire Format message set property **Output policy for xsi:type attributes**. For example, xsi:type attributes can be removed, output on all elements or output according to rules specified in the SOAP standard.

If validation is enabled for a WebSphere Message Broker message flow, the validation logic is sensitive to xsi:type attributes and uses them to modify the validation of the element. It will also validate the values of xsi:type attributes using the rules described in XML Schema Part 1: Structures on the World Wide Web Consortium (W3C) Web site.

There are several important points to remember when parsing and writing XML documents that contain xsi:type attributes.

- In order to detect and use xsi:type attributes, the message set must be namespace-enabled. To make a message set namespace-enabled, check the message set property **Use namespaces**.



- If the value of the `xsi:type` attribute contains a namespace prefix, the prefix will be expanded into a fully-qualified URI by the MRM XML parser. If the same `xsi:type` attribute is output later by the MRM XML writer, the same prefix will not automatically be used for the value. You can control the prefixes used on output using the **Namespace settings** list in the XML Wire Format message set properties. If no prefix is supplied, the XML writer will assign a default prefix.
- If the `xsi:type` attribute of an element does not resolve to a type in the model, then the behaviour depends on whether MRM validation is enabled. If not validating, the MRM will assume the type of the element is that declared in the model, and continue. If validating, a validation exception will be thrown.
- If MRM validation is enabled, any required `xsi:type` attributes must be present in the message tree at the point when validation is performed. An `xsi:type` attribute is required when its value is different from the data type of the element as defined in the message model (this most commonly occurs when using XML Schema type derivation).
  - If validation is being performed on an input message, then the MRM XML parser ensures that `xsi:type` attributes appear in the message tree, as described above.
  - If validation is being performed on an output message you must ensure that the correct `xsi:type` attributes appear in the message tree. You should ensure that any required `xsi:type` attributes are copied from input message tree to output message tree, or are explicitly created in the output message tree.
- If you are using simple types that are `xsd:unions` then an `xsi:type` attribute can be used to direct the MRM XML parser when resolving the union.
- If you are using WebSphere Message Broker Version 6.0 and have migrated from an earlier version which was not sensitive to `xsi:type` attributes, you might notice some changes of behavior. For example, in WebSphere Message Broker Version 6.0, `xsi:type` attributes are no longer treated as self-defining attributes, so they appear in the message tree with the name 'type' instead of '@type'. If your message flow logic is sensitive to `xsi:type` attributes in the message tree, you are recommended to change your message flow to comply with the new behavior. If you want to retain logic from an earlier version of WebSphere Message Broker in your message flows, this is described in Message flow migration notes.

For more information about `xsi:type` attributes, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

### Tagged/delimited string format

The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

The TDS physical format is designed to model messages that consist of text strings. Examples of TDS messages are those that conform to the ACORD AL3, EDIFACT, SWIFT, and X12 standards. The TDS physical format allows a high degree of flexibility when defining message formats, and is not restricted to modeling specific industry standards. You can therefore use the TDS format to model your own messages.

**TDS message characteristics:** There are a number of features of text string messages that are common across many formats. This is an overview of the main features that are supported by the TDS physical format:

**Tags** The text strings in the message can have a tag or a label preceding the data

value. The tag is a string that uniquely identifies the data value. The TDS format allows you to associate a tag with each element when you define the element.

### **Delimiters and tagged data separators**

The message can contain various special characters or strings in addition to the tags and text string data values. The TDS format supports a number of different types of special characters or strings.

Some messages have a special character or string that separates each data value from the next. In the TDS format this is known as a delimiter.

In formats that have a tag before each data value, the tag can be separated from its data value by a special character or string. In the TDS format this is known as a tag data separator.

### **Group indicators and terminators**

A message can be split into a number of substructures in a similar manner to a COBOL or C structure. You can model each of these substructures separately by defining groups, complex types or elements for each one.

A substructure can have a special character or string that indicates its start within the data. This is known in the TDS format as a group indicator.

A substructure can also have a special character or string that indicates its end in the data. In the TDS format, this is known as a group terminator.

A group indicator and group terminator can also be defined for the whole message. Group indicators and group terminators are optional for the message and each substructure.

### **Fixed length strings**

Some text strings within a message can be of fixed length, so a delimiter between each data value is not necessary. This is supported by the TDS format.

### **Fixed length tags**

Some tags can be defined as fixed length, so that a tag data separator is not necessary.

### **Separation types**

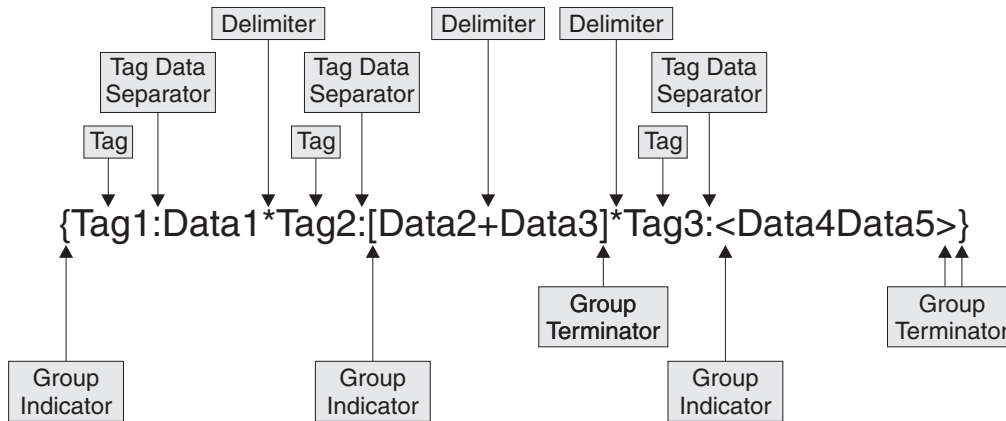
The TDS property that controls the way text strings are separated is Data Element Separation. It has several options that let you choose, for example, if tags are used, if strings lengths are fixed or variable, and what types of text strings are permitted.

The substructures within a message can use different types of data element separation and use different special characters. Therefore the TDS format allows you to define different types of data element separation and special characters for each complex type within the message.

### **Regular Expressions**

If you choose the Use data pattern option for Data Element Separation, you can use regular expressions to identify parts of the message data to be assigned to sub-fields. This is done by setting the regular expression in the Data Pattern property.

The diagram below shows an example data message with each of its components labeled.



- At the top level, each data value has a tag associated with it, each tag is separated from its data value using a tag data separator of colon (:), and the data values are separated from each other using the asterisk delimiter (\*).
- The group indicator for the message is the left brace ({}), and the group terminator is the right brace ({}).
- The data values Data2 and Data3 are in a substructure in which there are no tags, and each data element is separated from the next using the plus delimiter (+). The group indicator for this substructure is the left bracket ([) and the group terminator is the right bracket (]).
- The data values Data4 and Data5 are in a substructure in which the values are fixed length, and are therefore not separated by a delimiter. The group indicator for this substructure is the less than symbol (<) and the group terminator is the greater than symbol (>).

**Data element separation:** Data element separation defines how a TDS message is to be parsed.

Data element separation defines which method of identifying data elements is to be used and how the data elements are constructed. The different methods vary from full flexibility to fixed format, depending on how they are defined.

The four main types of data element separation are:

**Fixed length types**

Fixed length types are dependent on each element having a length. See “TDS format: Fixed length separation types” on page 52.

**Tagged separation types**

Tagged separation types are dependent on each element having tag prefix to identify it. See “TDS format: Tagged separation types” on page 53.

**Delimited separation types**

Delimited separation types use delimiters to identify the end of one data elements and the beginning of the next. See “TDS format: Delimited separation types” on page 55.

**Data pattern types**

Data pattern types use a regular expression to identify each element. See “TDS format: Data pattern separation types” on page 59.

There is a fifth category, which is different from the four described above:

### Undefined separation types

Undefined separation types contain no data elements. They are applicable to message content types (that is, embedded messages) only and should not be used for anything else. They use none of the TDS type-specific parameters other than Data Element Separation. See "Multipart messages" on page 28.

*TDS format: Fixed length separation types:* For fixed length separation types, each data value is a fixed length.

For fixed length data element separation types, all elements have a length or length reference, and are padded out to their full length in the bitstream. No tags or delimiters are used, and each data value follows the next immediately.

For example:

```
data1data200data30
```

The first element is length 5, the second is length 7 and the third is length 6. The padding character is "0".

*Fixed length type:* In fixed length type, all elements must have a length or length reference and must be written out to that full length. The elements must be presented in the right order, and all elements must be written in the bitstream. This includes all repeats of any repeating element (that is, the Maximum Occurrences must be written out for each element).

For example:

```
data10data2data2data2data300
```

The first element is length 6, the second is length 5 and repeats three times, and the third element is length 7. The padding character is "0".

*Applicable parameters:* The main parameters for this format are the Length or Length Reference of the element. All fields need to be padded out to their full length for the bitstream to be correctly specified to the parser.

Tag and delimiter parameters are ignored. Group indicators and terminators are observed, as they are of fixed length.

Default values are also required for each field that will not always be set, so that every field can be output even if it is not filled with data from the message.

*Fixed length AL3 type (Deprecated):* This separation type has been deprecated. ACORD AL3 support will be provided by a different method in a future release, at which time this separation type will be removed from service.

Fixed length AL3 types are similar to fixed length, but follow extra rules specified by the ACORD AL3 format regarding truncation and missing elements. If elements are missing from the end of an AL3 type, they can be truncated. They cannot be omitted from the middle of a bitstream. If a field is missing from the middle of the bitstream, that field is output as the appropriate length string of the "?" character.

*Applicable parameters:* The main parameters for this format are the Length or Length Reference of the element. All fields need to be padded out to their full length for the bitstream to be correctly specified to a parser.

Tag and delimiter parameters are ignored. Group Indicators and Terminators will be observed as they are of fixed length.

*TDS format: Tagged separation types:* For tagged separation types, each data value is preceded by a tag that is specified as an element property.

The Tag Data Separator, or specific Length of Tag parameter is used to determine where the tag ends and the data starts. Different methods are used by each separation type to determine the end of the data.

After considering these two parameters, this topic describes the following supported tagged separation types:

- “Tagged Delimited separation”
- “Tagged Fixed Length separation” on page 54
- “Tagged Encoded Length separation” on page 54

Tagged separation is a flexible format. The elements do not have to occur in a specific order. They do not all need to be present, and can be absent from any point in the message.

*Tag Data Separator and Tag Lengths:* Either Tag Data Separator and Length of Tag are used by all tagged separation types. But only one of these parameters can be set at the same time.

The point at which a tag ends and data starts can be determined by one of two methods. If the Tag Data Separator is set, then this character indicates where the data ends. For example, the string might be:

```
tag1:data1
```

where Tag Data Separator is :

However if the Tag Data Separator is not set and the Length of Tag field is set, then the tag is the specified length, and is immediately followed by the data. No separating character is required. For example, the string might be:

```
tag1data1
```

where Length of Tag is 4

*Tagged Delimited separation:* Tagged Delimited separation is a completely flexible format. The elements are not of specific lengths. Elements are separated by a predefined delimiter.

*Applicable parameters:* These parameters are used:

- Group Indicator indicates the start of a group or complex type.
- Group Terminator indicates the end of a group or complex type.
- Delimiter separates the data elements within a group or complex type.
- Tag for each element, indicates the tag needed to precede the data in that field.
- Either Tag Data Separator or Tag Length as described above.

*Examples:* If Tag Data Separator is set to :

```
{tag1:data1*tag2222222:data2*tag3333:data3}
```

where:

- Group Indicator is {

- Group Terminator is }
- Delimiter is \*
- Tag, defined for each element, is tag1, tag2222222, tag333 respectively

or, for example, if Length of Tag is set to 5  
{tag11data1\*tag22data2\*tag33data3}

where parameters are as above, except:

- Tag, defined for each element (fixed at 5 characters), is tag11, tag22, tag33 respectively

*Tagged Fixed Length separation:* Although Tagged Fixed Length separation is a flexible format, the data must be a specific length. This means that a delimiter is not needed to determine the end of each element.

*Applicable parameters:* These parameters are used:

- Group Indicator indicates the start of a group or complex type.
- Group Terminator indicates the end of a group or complex type.
- Tag for each element, indicates the tag needed to precede the data in that field.
- Either Length or Length Reference for each element, indicates the length of the data. (This value does not include the length of the tag.)
- Either Tag Data Separator or Tag Length as described above.

*Examples:* If Tag Data Separator is set to :

{tag1:data1tag2222222:data2000tag333:data300}

where:

- Group Indicator is {
- Group Terminator is }
- Delimiter is \*
- Tag, defined for each element, is tag1, tag2222222, tag333 respectively
- Length, defined for each element, is 5, 8, 7 respectively

or, for example, if Length of Tag is set to 5  
{tag11data1tag22data2000tag33data300}

where parameters are as above, except:

- Tag, defined for each element (fixed at 5 characters), is tag11, tag22, tag33 respectively

*Tagged Encoded Length separation:* This method has both a tag and a length field before the data. The length field indicates to the parser the length of the data following it.

The length of this length field is itself defined in the Length of Encoded Length parameter. Extra lengths to be added in this, such as the length of the field itself, is set in the Extra Chars in Encoded Length parameter.

These examples show how the values set in these parameters are applied:

- tagA007dataAAAtagB006dataBBtagC009dataCCCC  
If Length of Tag is 4, Length of Encoded Length is 3, Extra Chars in Encoded Length is 0, then in this bitstream, TagA is followed by the 3 character long

length field. This indicates that the following data (dataAAA) is 7 characters long. The next field, tagB is then considered, and so on.

- tagA012dataAAAAAtagB010dataBBBtagC016dataCCCCCCCC

If Length of Tag is 4, Length of Encoded Length is 3, Extra Chars in Encoded Length is 3, then in this bitstream, TagA is followed by the 3-character length field. This indicates that the following data, plus extra characters, is 12 characters long: length of the length field (3) + length of data (9) = 12. Therefore the length of the actual data is only 12-3 = 9. The next field, tagB is then considered, and so on. In each case the length given in the bitstream is 3 greater than the actual length of the data.

*Applicable parameters:* These parameters are used:

- Group Indicator indicates the start of a group or complex type.
- Group Terminator indicates the end of a group or complex type.
- Tag for each element, indicates the tag needed to precede the data in that field.
- Length of Encoded Length indicates the length of the length field in the bitstream.
- Extra Chars in Encoded Length indicates how many extra characters should be included in calculating the value for the length field in the bitstream.
- Either Tag Data Separator or Tag Length as described above.

*Examples:* If Tag Data Separator is set to :

```
{tag1111:008data1tag22222222:010data2AAtag3333:009data3A}
```

where:

- Group Indicator is {
- Group Terminator is }
- Length of Encoded Length is 3
- Extra Chars in Encoded Length is 3
- Tag, defined for each element, is tag1111, tag22222222, tag3333 respectively

or, for example, if Length of Tag is set to 5

```
{tag11008data1tag22010data2AAtag33009data3A}
```

where parameters are as above, except:

- Tag, defined for each element (fixed at 5 characters), is tag11, tag22, tag33 respectively

*TDS format: Delimited separation types:* For delimited separation types, a delimiter is used to separate data fields, but there are no tags present. The data fields need to be given in the correct order in the bitstream and elements cannot be omitted from the middle of the bitstream.

The All Elements Delimited separation type means that data fields are delimited by a pre-specified character or string. In this example, four data fields are separated by an asterisk (\*) delimiter:

```
data1*data2*data3*data4
```

Delimited separation types are restrictive in the ordering and presence of elements:

- The elements must be given in the order specified.
- No element can be omitted in the middle of a group or complex type, because the parser cannot determine this from the resulting bitstream.

- Elements can sometimes be absent from the end of a complex type or group.

After considering “Delimiter suppression and truncation rules,” this topic describes the following delimited separation types:

- “All Elements Delimited”
- “Variable Length Elements Delimited” on page 57

*Delimiter suppression and truncation rules:*

- Elements cannot be omitted from the middle of a group or complex type. An absent element results in the inclusion of a zero-length string.

For example, with all elements present, the string might be:

```
data1*data2*data3*data4
```

where Delimiter is \*

If data2 is missing, then the string would read:

```
data1**data3*data4
```

- It is possible to suppress the delimiters at the end of a string for absent elements. The Suppress Absent Element Delimiter determines whether this is done. If this parameter is set to End of Type then this can be done (with one exception, as will be seen later).

In this case, for the above example with data3 and data4 missing, the string would read:

```
data1*data2
```

That is, the delimiters have been suppressed from the end of this group or complex type.

- If Suppress Absent Element Delimiter is set to Never then delimiter suppression never takes place. The string would read:

```
data1*data2**
```

That is, the delimiters have to be present to indicate absent (zero-length) elements.

An exception to the above rule occurs in the case of the same delimiters being used at multiple levels in the model.

For example, you have a complex type or group with delimiter \* and this contains an element of another complex type (indicated by the element3 prefix on data fields in the example below), which also has delimiter \*. If both types use a delimited separation type, then with all elements present, you might have:

```
data1*data2*element3Data1*element3Data2*element3Data3*data4
```

If element3Data2 and element3Data3 were missing, and the delimiters were suppressed, then it would not be possible for the parser to determine which elements were missing.

Therefore in this case, you must override the Suppress Absent Element Delimiter parameter and write out all the delimiters to clearly define the message to the parser. So the string in this case must be:

```
data1*data2*element3Data1***data4
```

This restriction also applies where Group Indicators and Group Terminators use the same character strings as delimiters, otherwise the bitstream is not clear to the parser.

*All Elements Delimited:* In an All Elements Delimited separation type, all elements are separated by a delimiter; none are determined by their length. For example:

```
data1*data2*data3*data4*data5
```



where Delimiter is \*

*Applicable parameters:* These parameters are used:

- Group Indicator indicates the start of a group or complex type.
- Group Terminator indicates the end of a group or complex type.
- Delimiters separate the data elements within a group or complex type.
- Suppress Absent Element Delimiters indicates whether delimiter suppression is permitted (see below).

All Elements Delimited makes no use of tags or their associated parameters. Lengths are also ignored.

For example:

```
{data1*data22222*data3}
```

where:

- Group Indicator is {
- Group Terminator is }
- Delimiter is \*

*Repeating element rules:* If an element needs to be repeated in a type of All Elements Delimited, then the Repeating Element Delimiter (RED), is used to separate the repeated elements.

For example if data2 repeats 5 times:

```
data1*data2:data2:data2:data2:data2*data3*data4
```

where:

- Delimiter is \*
- Repeating Element Delimiter is :

If the Suppress Absent Element Delimiters field is set to End of Type, then you can use delimiter suppression. So if, in the above example, only the first data2 element was present, then the bitstream reads:

```
data1*data2*data3*data4
```

However, if the Suppress Absent Element Delimiters field is set to Never, then the bitstream reads:

```
data1*data2::::*data3*data4
```

If the delimiter and RED match, then two delimiters are output to indicate that the repeat is ending. So if the delimiter and RED are \*, then the bitstream reads:

```
data1*data2**data3*data4
```

*Variable Length Elements Delimited:* In a Variable Elements Delimited separation type (also known as Delimited and Fixed separation type) some elements are determined by their length, while others are delimited. It is really a combination of a delimited and a fixed length format, and follows rules associated with both. Lengths can be given and used, but they are not mandatory.

- If a length is present for an element, then it is used, and a delimiter is not needed to terminate that element. The element must be padded to the correct length, and cannot exceed that length.
- If no length is given, then the delimiter is required.

A Variable Elements Delimited separation type containing only variable length elements resembles an All Elements Delimited type. If it contains only fixed length elements, it resembles a Fixed Length type.

For example:

```
data1*data2*data3*data4000data5
```

where:

- Delimiter is \*
- data4 has a length of 8

*Applicable parameters:* These parameters are used:

- Group Indicator indicates the start of a group or complex type.
- Group Terminator indicates the end of a group or complex type.
- Delimiters separate the data elements within a group or complex type.
- Suppress Absent Element Delimiters indicates whether delimiter suppression is permitted.
- (Optionally) one of Length or Length Reference indicates the length of an element. If an element has a length, then this length is used. As the length of this element is known, it is then not necessary to output a delimiter after it. If the length is not known, then an element is required.

In this example, the fourth field (containing data4) is of fixed length 8 and its padding character is 0:

```
{data1*data22222*data3*data4000data5}
```

where:

- Group Indicator is {
- Group Terminator is }
- Delimiter is \*

*Repeating element rules:* The action of a repeating element in a Variable Elements Delimited environment is dependent on the minimum and maximum number of repeats and whether the element has a length.

*Delimited Element Repeating:* If a delimited element (that is, an element with no length) is repeated, then a Repeating Element Delimiter (RED) is required and the rules for All Elements Delimited are followed. A delimiter is therefore required after the last repeat. Delimiter suppression of this repeat can also occur.

For example, if data2 is repeating:

```
data1*data2:data2:data2:data2:data2:data2*data3*data4000data5
```

where:

- Delimiter is \*
- Repeating Element Delimiter is :
- data4 has a fixed length of 8

If the Suppress Absent Element Delimiters field is set to End of Type then you can use delimiter suppression.

If in the above example only the first data2 is present:



No delimiters or tags, other than those coded as part of the regular expression pattern, are used in the bitstream. See "Using regular expressions to parse data elements" on page 695 for an explanation of pattern matching.

For example, if the first three Data Pattern properties are, respectively:

- [A-Z] (1,3)
- [0-9]+
- [a-z]\*

and the message data is:

DT31758934information for you

Then, in this example:

- First data element = DT
- Second data element = 31758934
- Third data element = information

The first data pattern means "from one to three characters in the range A to Z", the second means "one or more characters in the range 0 to 9" and the third means "zero or more characters in the range a to z". Note how each element's data was terminated by the first character that did not match the element's Data Pattern.

## Performance issues

The parsing required in Data Pattern separation type is the slowest of all the different separation types because of its complexity.

You should therefore only use Data Pattern separation type when no other type will model the message. It should not be used to replace, for example, Fixed Length separation type.

*Applicable parameters:* Only one parameter is used:

- Data Pattern for each element, indicates the regular expression used for string matching.

**Tagged/delimited string format - Model integrity:** When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. This is necessary to avoid any discrepancies when processing a message within the specified model.

*Rules of TDS physical format properties:* Restrictions to message formats are checked. These will follow the rules specified in "TDS message model integrity" on page 692. The majority of rules are applied for at least one of these reasons:

### Rules for message definition

Some rules are necessary for the message to be completely defined.

For example, in a Fixed Length separation type all elements must have some length defined, either directly or by using a Length Reference. Without this information, it is impossible to tell in the message bitstream where one data element ends and the next starts.

### Rules for nesting

Nesting rules relate to which separation types can be nested inside each other.

Such rules are applied when an element of a complex type is present inside another complex type. An example of this would be that it is not possible to have a Tagged Delimited separation type inside a Fixed Length type. Since a Tagged Delimited separation type is of variable length, the parent Fixed Length type would be unable to tell where that particular element ended, as there would be no length provided. Therefore the message could not be processed.

### **Rules linking to the logical model**

There are also rules linking TDS to the logical model.

These occur where a group composition or group content validation cannot be used in conjunction with a particular separation type. Again this is for message integrity. For example, a separation type of All Elements Delimited cannot have a group composition of Open, as there is no information as to what the extra elements represent and where they are in the bitstream.

**Tagged/delimited string format - NULL handling:** NULL handling dictates the way in which the MRM parser for TDS messages handles elements that have been set to Null.

The rules for whether nulls are permitted are described in “TDS Null handling options” on page 691.

*Null properties:* The element properties Encoding Null and Encoding Null Value control how null handling is represented for individual elements.

You can select the Encoding Null property from the three enumerated values NULLPadFill, NULLLogicalValue, and NULLLiteralValue. The use of the Encoding Null Value property is dependent on the value that you select for the Encoding Null property.

NULL values are not defined for schema `xsd:hexBinary` simple types. The properties Encoding Null and Encoding Null Value are therefore not set for `xsd:hexBinary` types.

NULL values for schema `boolean` simple types are defined at the message set level. The message set property Boolean Null Representation specifies the value to be used for Boolean Null representation.

**Tagged/delimited string format - Multipart messages:** The Tagged/Delimited String Format (TDS) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

The SWIFT, X12 and EDIFACT messaging standards can all be modeled using the Message Identity technique.

Versions of the TDS physical format prior to Version 6.0 included embedded message identification by Message Key which worked in a similar manner to Message Identity, but which applied to TDS only. The Message Key technique has been deprecated and is superseded by Message Identity. Warning task list messages are issued if the use of Message Key is detected, and a task list Quick Fix may be selected to create the equivalent Message Identity automatically. You must continue to use Message Key if the MRM parser that you are deploying to is Version 5.0 or older.

**Tagged/delimited string format - Data conversion:** All TDS message data apart from binary types are handled as strings. All string data is therefore subject to CCSID conversion only. This includes the special characters used as delimiters, data separators, and so on.

**Tagged/delimited string format: relationship to the logical model:** There are some restrictions between TDS separation types and logical model properties such as group composition and group content validation. The rules governing these options are explained in “Restrictions for nesting complex types” on page 694.

These rules exist to ensure the integrity of the message. A combination of separation type and group composition or group content validation must not lead to a message that is unclear to a TDS parser.

## Default values

In TDS, *Default* values are only observed by fixed length elements:

Separation Type	Use of Default values
Tagged Delimited Tagged Fixed Length Tagged Encoded Length All Elements Delimited Data Pattern	Default values are never observed.
Fixed Length Fixed Length AL3	Default values are observed on output by all elements. An absent element that has no Default value defined, will cause an error on writing.
Variable Length Elements Delimited	Default values are only observed by fixed length elements on output. Absent fixed length values must have a Default value available to them. An absent element that has no Default value defined, will cause an error on writing.

## Simple types – lists and unions

Lists and unions are XML-specific concepts. An element or attribute of a simple type that is a list or a union will cause a task list warning if a TDS physical format is present in the message set. The user can choose whether to make this an error, warning, or information by editing the Validation preferences. If a dictionary is generated from the message set, and an attempt is made to parse a TDS message defined to contain such elements or attributes, a runtime error will occur.

## Min Occurs and Max Occurs

The logical properties Min Occurs and Max Occurs specify the permitted number of occurrences of an element or group in a message. They are used when parsing and writing messages, and when validating the content of a message.

When parsing and writing, the exact interpretation of these properties depends on the Data Element Separation property of the parent complex type or group as shown in the table below.

When validating, Min Occurs and Max Occurs are both used to check that the content of the message tree matches the model.

Separation type	Interpretation of Min Occurs and Max Occurs
Tagged Delimited Tagged Fixed Length Tagged Encoded Length	<p>Min Occurs and Max Occurs are effectively ignored when parsing and writing. When parsing, the number of occurrences is identified by the tags in the message. When writing, the writer outputs all occurrences in the message tree.</p> <ul style="list-style-type: none"> <li>• A varying number of occurrences (Min Occurs &lt;&gt; Max Occurs) is allowed.</li> <li>• Optional occurrence (Min Occurs = 0) is allowed.</li> <li>• Always absent (Max Occurs = 0) is allowed.</li> <li>• An unbounded number of occurrences (Max Occurs = -1) is allowed.</li> </ul>
All Elements Delimited	<p>Max Occurs only is used when parsing and writing, in conjunction with the element's Repeating Element Delimiter property, and the parent type's Suppress Absent Element Delimiters property.</p> <p>A varying number of occurrences (Min Occurs &lt;&gt; Max Occurs) is allowed if Suppress Absent Element Delimiters is set to End of Type.</p> <ul style="list-style-type: none"> <li>• If the Delimiter is different to the Repeating Element Delimiter then the Delimiter will signify the end of the occurrences.</li> <li>• If the Delimiter is the same as the Repeating Element Delimiter then an empty repeat signifies the end of the occurrences.</li> <li>• In both these cases, Max Occurs is the maximum number of repeats that are expected.</li> </ul> <p>If Suppress Absent Element Delimiters is Never, then all occurrences are expected when parsing, and output when writing, although parsing will accept elements being absent.</p> <p>Optional occurrence (Min Occurs = 0) is ignored and a delimiter is still expected when parsing, and output when writing.</p> <p>Always absent (Max Occurs = 0) is allowed. No delimiter is expected when parsing, nor output when writing.</p> <p>An unbounded number of occurrences (Max Occurs = -1) is only allowed if the Repeating Element Delimiter is different from the Delimiter. The repeats must be terminated by the delimiter, or a containing group's Group Terminator or Delimiter, or by the end of the message bitstream. On writing, the writer outputs all occurrences in the message tree.</p>
Fixed Length Fixed Length AL3	<p>Max Occurs only is used when parsing and writing. In general, Max Occurs occurrences are expected when parsing, and Max Occurs occurrences are output when writing with default values used for missing elements.</p> <p>A varying number of occurrences (Min Occurs &lt;&gt; Max Occurs) is ignored, Max Occurs is assumed.</p> <p>Optional occurrence (Min Occurs = 0) is ignored, Max Occurs is assumed.</p> <p>Always absent (Max Occurs = 0) is allowed.</p> <p>Fixed Length only. An unbounded number of occurrences (Max Occurs = -1) is allowed if the element or group is the last child in its parent group, and the group is terminated by a Group Terminator or a containing group's Group Terminator or Delimiter or by the end of the message bit stream. On writing, the writer outputs all occurrences in the message tree, if this number is less than Min Occurs then additional default values are written.</p>

Separation type	Interpretation of Min Occurs and Max Occurs
Variable Length Elements Delimited	<p>For fixed length simple elements, the rules for Fixed Length separation above are followed with two differences.</p> <ol style="list-style-type: none"> <li>1. A varying number of occurrences (Min Occurs &lt;&gt; Max Occurs) is allowed, the end of the occurrences being signified by an extra delimiter.</li> <li>2. An unbounded number of occurrences (Max Occurs = -1) is allowed, the end of the occurrences being signified by an extra delimiter. On writing, the writer outputs all occurrences in the message tree, followed by an extra delimiter.</li> </ol> <p>For variable length simple elements, all complex elements and groups, the rules for All Elements Delimited above are followed.</p>
Data Pattern	<p>Min Occurs and Max Occurs are effectively ignored when parsing and writing. When parsing, the pattern is matched as many times as possible. When writing, the writer outputs all occurrences in the message tree. Note that on parsing, if the data pattern permits a zero length match, and a zero length match occurs, an element is added to the message tree and the matching terminates to prevent an infinite loop.</p> <p>A varying number of occurrences (Min Occurs &lt;&gt; Max Occurs) is allowed.</p> <p>Optional occurrence (Min Occurs = 0) is allowed. Always absent (Max Occurs = 0) is allowed.</p> <p>An unbounded number of occurrences (Max Occurs = -1) is allowed.</p>

## Ways to create message definitions

Once you have created a message set, you need to populate your message set with message definitions. You would typically do this by either:

- Importing application message formats described by XML DTD, XML Schema, WSDL definitions, IBM supplied messages, C structures or COBOL structures, which will create and populate message definition files.
- Creating empty message definition files and then creating your messages using the Message Definition Editor.

Both these alternatives are accessed using the New Message Definition File wizard. Additionally you can import application message formats using the `mqsicreatemsgdefs` or `mqsicreatemsgdefsfromwsdl` command line utilities. `mqsicreatemsgdefs` has a bulk import capability, but `mqsicreatemsgdefsfromwsdl` will import only one WSDL definition at a time.

### Import from other model representations to create message definitions

One way to populate your message set with message definitions is to import existing application message formats. You can import the following message formats into a message set:

- C header files
- COBOL copybooks
- XML DTD files
- XML Schema files
- WSDL definitions
- IBM supplied messages

When you import one of these formats, a new message model is created consisting of the elements, attributes, groups and types needed to represent your message



format. You choose the name of the message definition file; if it already exists the content is deleted and recreated as part of the import operation.

The new message model created can consist of both logical and physical information, if appropriate physical formats exist in the message set at the time of the import.

All imports are performed using either the New Message Definition File wizard or the `mqsicreatemsgdefs` command line utility. The command line utility allows you to import several message format files of varying formats in a single operation, and allows you to create a new message set (based on an existing message set) into which the message definition files will be placed.

**Client application access to messages:** Client applications also need access to message definitions to be able to construct messages they send, and interpret messages that they receive.

- If the message formats have been imported from C or COBOL structures using the workbench, your applications can continue to use the same C and COBOL data structures that were imported to create the message dictionary that will be used by the brokers.
- If the messages are self-defining XML, the client applications must construct valid messages using the structures that can be understood by the recipients of the message.

**Import from C header files to create message definitions:** You can populate your message set with message definitions by importing C header files, using either the New Message Definition File wizard or the `mqsicreatemsgdefs` command line utility.

Each C header file you import results in a new message definition file. The root name of the message definition file defaults to the root name of the C header file, but the New Message Definition File wizard allows you to choose a different root file name.

If the message definition file already exists, you must have permitted overwriting to occur for the import to proceed, in which case the existing content is deleted and recreated.

By default, all message definition files created as a result of C header file import belong to the `noTarget` XML namespace and so will reside in the (default) location in your workspace. This default namespace can be overridden by specifying a target namespace. For reasons why you might want to do this, see “Namespaces with non-XML messages” on page 37.

In your C header file there will typically be one or more C structures. You can select which of these structures to import. The import operation then imports those structures, plus any others they require. All imported structures are converted into the equivalent elements, groups and types in the message definition file.

You can also specify which of the selected structures are to be messages in the message definition file. The import operation creates a message and corresponding global element in the message definition file for each structure you specify. If you do not specify that messages are to be created, you must create them manually using the Message Definition Editor after the import has completed.

When you are importing using the New Message Definition File wizard you can only import one C header file with each import operation. If importing using the command line utility you can import several C header files in each import operation.

If your C header file needs any other header files in order to compile successfully, you must provide these and specify their location, because a compilation of your header file is performed as part of the import operation.

A report file is created during the import operation. This is located by default in the log folder of the message set. By default it takes the name of the message definition file, with `.report.txt` appended.

*Physical information:* As well as creating logical information, the import can also create physical information. If the message set contains any Custom Wire Format (CWF) physical formats, then the physical format properties for all CWF layers are populated. If the message set does not contain any CWF physical formats, only logical information is created. Also, if you import from the command line, only logical information is created in the new message set by default. If you want physical information created as well, see “Importing from the command line” on page 127 for details.

XML and TDS physical format properties are not populated and so take default values.

If you have one or more TDS layers, the import can cause entries in the task list, warning you that certain TDS properties must be set if the C structures you have imported were to appear in a TDS message.

If the TDS physical format is not applicable to your C structures, you can ignore these task list entries because they are just warnings, they will not prevent your model being generated in another form, such as a message dictionary.

Because physical information is created, the application target environment (platform and compiler) is important because it governs the way that, for example, integers appear in the message. You can specify environment specific information as part of the import operation, and the necessary properties will be set accordingly. There is a range of environments supported; if your environment is not shown, you should choose the closest match and review the created physical information using the Message Definition Editor after the import has completed.

*Command line invocation:* The `mqsicreatemsgdefs` command line utility allows you to import several C header files in a single operation. All the C header files must be placed in the same directory and the directory location passed as a parameter to the utility.

You provide the necessary environment-specific information, and include file location information using an XML file. This must contain an XML element called `<C>` which holds one set of information that applies to all C header files imported during an invocation of the utility. A default XML file called `mqsicreatemsgdefs.xml` is supplied. If you want to apply different sets of information to different header files, you need to create multiple XML files and run the utility multiple times.

The `mqsicreatemsgdefs` utility also allows you to create a new message set into which the message definition files are placed, as part of the import operation. You

can also choose to base this new message set on an existing message set. This facility enables you to prepare an empty message set containing a CWF physical format and message set level CWF properties already populated, which then gets copied into the message set created by the import.

**Import from COBOL copybooks to create message definitions:** You can populate your message set with message definitions by importing COBOL copybook files, using either the New Message Definition File wizard or the `mqsicreatemsgdefs` command line utility.

Each COBOL copybook you import results in a new message definition file. The root name of the message definition file defaults to the root name of the COBOL copybook file, but the New Message Definition File wizard allows you to choose a different root file name.

If the message definition file already exists, you must have permitted overwriting to occur for the import to proceed, in which case the existing content is deleted and recreated.

By default, all message definition files created as a result of COBOL copybook file import belong to the `noTarget` XML namespace and so will reside in the (default) location in your workspace. This default namespace can be overridden by specifying a target namespace. For reasons why you might want to do this, see “Namespaces with non-XML messages” on page 37.

In your COBOL copybook file there will typically be one or more level 01 structures. You can select which of these structures to import. The import operation then imports those structures, plus any others they require. All imported structures are converted into the equivalent elements, groups and types in the message definition file.

You can also specify which of the selected level 01 structures are to be messages in the message definition file. The import operation creates a message and corresponding global element in the message definition file for each structure you specify. If you do not specify that messages are to be created, you must create them manually using the Message Definition Editor after the import has completed.

When you are importing using the New Message Definition File wizard you can only import one COBOL copybook file with each import operation. If importing using the command line utility you can import several COBOL copybook files in each import operation.

If your COBOL copybook file needs any other copybooks in order to compile successfully, you must provide these in the same directory, because a compilation of your copybook is performed as part of the import operation.

A report file is created during the import operation. This is located by default in the log folder of the message set. By default it takes the name of the message definition file, with `.report.txt` appended.

*Physical information:* As well as creating logical information, the import can also create physical information. If the message set contains any Custom Wire Format (CWF) physical formats, then the physical format properties for all CWF layers are populated. If the message set does not contain any CWF physical formats, only logical information is created. Also, if you import from the command line, only

logical information is created in the new message set by default. If you want physical information created as well, see “Importing from the command line” on page 127 for details.

XML and TDS physical format properties are not populated and so take default values.

If you have one or more TDS layers, the import can cause entries in the task list, warning you that certain TDS properties must be set if the COBOL structures you have imported were to appear in a TDS message.

If the TDS physical format is not applicable to your COBOL structures, you can ignore these task list entries because they are just warnings, they will not prevent your model being generated in another form, such as a message dictionary.

Because physical information is created, the application target environment (platform and compiler) is important because it governs the way that, for example, integers appear in the message. You can specify environment specific information as part of the import operation, and the necessary properties will be set accordingly. There is a range of environments supported; if your environment is not shown, you should choose the closest match and review the created physical information using the Message Definition Editor after the import has completed.

*Command line invocation:* The `mqsicreatemsgdefs` command line utility allows you to import several COBOL files in a single operation. All the COBOL copybook files must be in single directory and the directory location passed as a parameter to the utility.

You provide the necessary environment specific information using an XML file. This must contain an XML element called `<COBOL>` which holds one set of environment specific information that applies to all COBOL copybook files imported during an invocation of the utility. A default XML file called `mqsicreatemsgdefs.xml` is supplied. If you want to apply different sets of information to different copybooks, you need to create multiple XML files and run the utility multiple times.

The `mqsicreatemsgdefs` utility also allows you to create a new message set into which the message definition files are placed, as part of the import operation. You can also choose to base the message set created on an existing message set. This facility enables you to prepare an empty message set containing a CWF physical format and pre-populated message set level CWF properties, which are then copied into the message set created by the import.

**Import from DTDs to create message definitions:** You can populate a message set with message definitions by importing DTD files, using either the New Message Definition File wizard or the `mqsicreatemsgdefs` command line utility.

Each XML DTD file you import results in a new message definition file within the message set. The root name of the message definition file defaults to the root name of the XML DTD file, but the New Message Definition File wizard allows you to choose a different root file name.

If the message definition file already exists, you must have permitted overwriting to occur for the import to proceed, in which case the existing content is deleted and recreated.

All message definition files created as a result of DTD file import belong to the noTarget XML namespace and so will reside in the (default) location in your workspace.

A report file is created during the import operation. This is located by default in the log folder of the message set. By default it takes the name of the message definition file, with .report.txt appended.

*Import using the New Message Definition File wizard:* When you import using the New Message Definition File wizard, you can specify which of the global elements within the imported XML DTD file are to be messages within the message definition file.

You can only import one XML DTD file with each import operation. If your XML Schema file references other XML DTD files, with import or include elements, these XML DTD files must be imported with a separate import operation into the same message set.

*Import using the command line:* When you import using the command line you have the option of either creating no messages or creating a message for each global element within the imported XML DTD file. The import operation creates a message and corresponding global element in the message definition file for each global element you specify. If you do not specify that messages are to be created, you must create them manually using the message definition editor after the import has completed.

You can import several XML DTD files in each import operation.

*Physical information:* As well as creating logical information, the import can also create physical information. If the message set contains any XML wire format physical formats, then the physical format properties for all XML Wire Format layers is populated. If the message set does not contain any XML physical formats, only logical information is created. Also, if you import from the command line, only logical information is created in the new message set by default. If you want physical information created as well, see “Importing from the command line” on page 127 for details.

CWF and TDS physical format properties are not populated and so take default values.

If you have one or more CWF or TDS layers, the import can cause entries in the task list, warning you that certain CWF or TDS properties must be set if the XML structures you have imported were to appear in a CWF or TDS message.

If the CWF or TDS physical formats are not applicable to your XML structures, you can ignore these task list entries because they are just warnings, they will not prevent your model being generated in another form, such as a message dictionary.

*Command line invocation:* The mqsicreatemsgdefs command line utility allows you to import several XML DTD files in a single operation. All the XML DTD files must be in single directory and the directory location passed as a parameter to the utility.

The mqsicreatemsgdefs utility also allows you to create a new message set into which the message definition files are placed, as part of the import operation. You can also choose to base the message set created on an existing message set. This

facility enables you to prepare an empty message set containing a XML physical format and pre-populated message set level XML properties, which are then copied into the message set created by the import.

*Further information about XML DTDs:* For details about XML DTDs, see the World Wide Web Consortium (W3C) Web site.

**Import from XML schemas to create message definitions:** You can populate a message set with message definitions by importing XML Schema files, using either the New Message Definition File wizard or the `mqsicreatemsgdefs` command line utility.

Each XML Schema file you import results in a new message definition file within the message set. The root name of the message definition file defaults to the root name of the XML Schema file, but the New Message Definition File wizard allows you to choose a different root file name.

If the message definition file already exists, you must have permitted overwriting to occur for the import to proceed, in which case the existing content is deleted and recreated.

The namespace to which the message definition file created belongs depends on whether namespaces have been enabled for the message set.

- If namespaces have been enabled the message definition file belongs to the target namespace of the XML Schema file being imported.
- If namespaces have not been enabled for the message set the message definition file belongs to the `noTarget` XML namespace irrespective of the target namespace of the imported XML Schema file and so will reside in the (default) location in your workspace. The implications of importing into a message set with namespaces disabled are described in “Importing XML Schema into message sets with namespaces disabled” on page 71.

A report file is created during the import operation. This is located by default in the log folder of the message set. By default it takes the name of the message definition file, with `.report.txt` appended.

*Import using the New Message Definition File wizard:* When you import using the New Message Definition File wizard, you can specify which of the global elements within the imported XML Schema file are to be messages within the message definition file.

You can only import one XML Schema file with each import operation. If your XML Schema file references other XML Schema files, with `import` or `include` elements, these XML Schema files must be imported with a separate import operation into the same message set.

*Import using the command line:* When you import using the command line you have the option of either creating no messages or creating a message for each global element within the imported XML Schema file. The import operation creates a message and corresponding global element in the message definition file for each global element you specify. If you do not specify that messages are to be created, you must create them manually using the message definition editor after the import has completed.

You can import several XML Schema files in each import operation.

*Physical information:* As well as creating logical information, the import can also create physical information. If the message set contains any XML wire format physical formats, then the physical format properties for all XML Wire Format layers is populated. If the message set does not contain any XML physical formats, only logical information is created. Also, if you import from the command line, only logical information is created in the new message set by default. If you want physical information created as well, see “Importing from the command line” on page 127 for details.

CWF and TDS physical format properties are not populated and so take default values.

If you have one or more CWF or TDS layers, the import can cause entries in the task list, warning you that certain CWF or TDS properties must be set if the XML structures you have imported were to appear in a CWF or TDS message.

If the CWF or TDS physical formats are not applicable to your XML structures, you can ignore these task list entries because they are just warnings, they will not prevent your model being generated in another form, such as a message dictionary.

*Command line invocation:* The `mqsicreatemsgdefs` command line utility allows you to import several XML Schema files in a single operation. All the XML Schema files must be in single directory and the directory location passed as a parameter to the utility.

When you import into a message set for which namespaces are not enabled, the action to take for unsupported constructs can be specified using an XML options file. This must contain an XML element called `<XSD_NO_NS>` which holds a set of information that applies to all XML Schema files imported during an invocation of the utility. A default XML options file is supplied called `mqsicreatemsgdefs.xml`. If you wish to apply different sets of information to different XML Schema files, you need to create multiple XML files and run the utility multiple times.

There are 2 other options that you can specify in the `<XSD_NO_NS>` element in the XML options file when importing into a message set for which namespaces are not enabled:

- The prefix to use for the `http://www.w3.org/2001/XMLSchema-instance` namespace; the default is `xsi`.
- Additional namespace URI and prefix pairs.

The `mqsicreatemsgdefs` utility also allows you to create a new message set into which the message definition files are placed, as part of the import operation. You can also choose to base the message set created on an existing message set. This facility enables you to prepare an empty message set containing a XML physical format and pre-populated message set level XML properties, which are then copied into the message set created by the import.

*Further information about XML Schema:* For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

*Importing XML Schema into message sets with namespaces disabled:* You can import an XML Schema file with a target namespace even if the message set does not have namespaces enabled. This is useful when the message set is to be used with message flows running on WebSphere MQ Integrator Broker Version 2.1, or that

originated on WebSphere MQ Integrator Broker Version 2.1. Such message flows will not be namespace aware, and will require all definitions in the message set to be in the noTarget namespace.

When importing an XML Schema file with a target namespace into a message set for which namespaces have not been enabled, the created message definition file is placed in the noTarget XML namespace. In some cases this can lead to name conflicts if global constructs have the same name in different namespaces in the XML Schema files imported into the same message set. These will cause error entries in the task list which you must resolve before generating the model in another format, such as a message dictionary.

As all the message definition files are in the noTarget XML namespace, the namespace information associated with the XML schema file is lost. However, the importer provides a limited form of namespace support by prefixing the XML names in the XML Wire Format layers with a namespace prefix. To allow this namespace support to work, an XML Schema file imported must specify an xmlns attribute with a non-empty prefix for the target namespace of the XML Schema file. This is the prefix used in the XML names in the XML Wire Format layers.

This means that you cannot specify the target namespace of the XML file as the default namespace. Each namespace in the XML schema files must use a unique prefix and the same namespace must always use the same prefix. Any XML instance documents, against which you match any of the forms generated from the model, must also use the same prefixes for the namespaces.

The XML Schema importer creates a number of optional attributes in an attribute group to represent namespace information. This attribute group is referenced by the type of any message. An attribute is created to represent the schema location of the XML schema file, and an attribute is created to represent the mapping of the prefix to the <http://www.w3.org/2001/XMLSchema-instance> namespace. An attribute is also created for each xmlns attribute in the XML Schema document. When importing using the Message Definition File wizard the prefix <http://www.w3.org/2001/XMLSchema-instance> namespace can be changed and additional namespace URI/prefix pairs added using the last panel of the Message Definition File wizard. When you use the mqscreatemsgdefs command line utility, the same modifications can be made using the XML options file.

*Further information about XML Schema:* For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

**Import from WSDL files to create message definitions:** You can add messages to a message set by importing WSDL files using either the New Message Definition File wizard or the mqscreatemsgdefsfromwsdl command line utility.

Each WSDL file you import results in one or more new message definition file within the message set. A new message definition file is created for each namespace defined for the message set. The name of the message definition file defaults to the name of the WSDL file, but the New Message Definition File wizard will allow you to choose a different file name.

If the message definition file already exists, you must have permitted overwriting to occur for the import to proceed, in which case the existing content is deleted and recreated.



Namespaces must be enabled in the message set that you are importing the WSDL file into.

A report file is generated during the import operation which will allow you to examine what occurred during the import process and check any errors that could have resulted.

## Import using the New Message Definition File wizard

When you import using the New Message Definition File wizard, you can only specify one WSDL definition for each import operation. A WSDL definition could be held as one or more WSDL files and these will all be imported as a result of importing the definition.

## Import using the command line

The WSDL command line importer (`mqsicreatemsgdefsfromwsdl`) can create a new message set or update an existing one. If the import succeeds, new message definition files are added to the message set. If the message set project exists, it must be namespace-enabled. If the project does not exist, a new namespace-enabled project is created.

The `mqsicreatemsgdefsfromwsdl` command allows you to import one WSDL definition in a single operation.

The `mqsicreatemsgdefsfromwsdl` command copies the WSDL files it needs into the workspace before the import runs. These are the top level WSDL files and any imports are resolved using an absolute or relative location. The files are copied under the specified message set in a folder called `importFiles` and are not removed after the import, allowing the user to subsequently update or run validation on them in the workbench.

## Physical information

If physical formats are required, they must be added to the message set prior to importing the WSDL definition. If the message set contains any physical formats, then the physical format properties for all physical format layers are populated.

If the message set does not contain any physical formats, only logical information is created. Also, if you import from the command line, only logical information is created in the new message set by default.

*Relationship of WSDL to Message Model:* The format of the SOAP messages is described in a WSDL definition, so the broker must be able to map between that WSDL and the message model it uses to parse those messages at runtime. The purpose of this topic is to highlight the key relationships between WSDL and the message model.

The physical parts of the WSDL definition are not represented in the broker model, so you only need to consider the WSDL operation, message and part definitions. Starting with the lower level building blocks, a WSDL definition describes:

- a number of logical messages and their constituent parts which are themselves defined in terms of XML schema. The part definitions are imported directly into the message model to create the corresponding element and type definitions. The definitions are allocated to message definition files according to the target

namespace of their schema definition. If there is no `targetNamespace` defined on the `<xsd:schema>` element, then the resulting elements and types have no namespace.

- a number of operations comprising the WSDL portType or interface. The operations define the business payload for the SOAP messages at runtime. Broker messages are created for each possible payload.
- a SOAP version and encoding style. Currently both the WSDL importer and generator assume the use of SOAP 1.1. This is because WSDL 1.1 only has a standard binding for SOAP 1.1. Note that there is no reason that WSDL 1.1 cannot define a web service that uses SOAP 1.2 – its simply that there isn't a standard way of doing this. If your web service does use SOAP 1.2 then you may need to manually remove the SOAP 1.1 definitions and import the SOAP 1.2 definitions.

## Message categories

The WSDL generator uses message categories to define the required WSDL operations. However message categories have no runtime significance in the broker message model and are not created when WSDL is imported.

The relationship of a Message Category to the broker message model depends upon the WSDL style:

- WSDL `rpc-style` offers a function-style approach to presenting the data being transmitted. An `rpc-style` WSDL may use either 'SOAP encoding' or 'literal encoding'.

The single child of the `<soap:Body>` represents the `rpc` (function) call and is referred to as the `rpc` wrapper or accessor. The name of the `rpc` accessor is given by the WSDL operation name and its namespace is given by the namespace attribute on the `<soap:body>` definition, which is only used for `rpc-style`. The children of the `rpc` accessor represent the parameters of the `rpc` call. The names of the child elements are given by the part names from the corresponding WSDL message definition. These are local to the `rpc` accessor and have no namespace.

SOAP headers and faults can be used in conjunction with `rpc-style` messages, but these do not map to `rpc-style` parameters and are required to use the document style by the WS-I Basic Profile.

- WSDL `document-style` offers a message-style approach to presenting the data being transmitted.

The messages exchanged are completely described by XML Schema (only literal encoding is allowed). No parts of the SOAP message are derived from WSDL objects such as operation and part names.

The WS-I Basic Profile specifies that a document-style message must have no more than one body part. The `SOAP-ENV:Header` element can have multiple child elements (header blocks). The fault detail message is defined by a single WSDL message part.

In each case the element schema is defined by the element referenced in the WSDL message part definition and belongs to the `targetNamespace` of the `<schema>` that defines the element.

## Resulting message model

The resulting model allows the user to parse incoming SOAP messages using the MRM XML parser where the message type would be `Envelope`. The message model for the SOAP envelope defines the outer SOAP wrapper with its constituent Header and Body sections and a number of attachment points where the various

business payloads can appear. These attachment points are defined with composition 'message', allowing the broker messages created by the WSDL importer or generator to appear at these points.

The table shows the attachment points and the possible attachments at each point.

Attachment Point	WSDL Style	
	rpc	document
Envelope.Body	single operation accessor	should have single child
Envelope.Header	may have multiple children including duplicates	may have multiple children including duplicates
Envelope.Body.Fault.detail	should have single child	should have single child

**Import from IBM supplied messages to create message definitions:** You can add messages to a message set by importing IBM supplied messages using the New Message Definition File wizard.

Each IBM supplied message you import results in one or more new message definition file within the message set. A new message definition file is created for each namespace defined for the message set. The name of the message definition file defaults to the name of the IBM supplied message, but the New Message Definition File wizard will allow you to choose a different file name.

When you import using the New Message Definition File wizard, you can only specify one IBM supplied message definition for each import operation.

If the message definition file already exists, you must have permitted overwriting to occur for the import to proceed, in which case the existing content is deleted and recreated.

A report file is generated during the import operation which will allow you to examine what occurred during the import process and check any errors that could have resulted.

## Generate model representations

After you have created and populated a message set, you can generate a message model in several different representations for use by your applications:

- A message dictionary for deployment to WebSphere Message Broker.
- A W3C XML Schema for use by an application building or processing XML messages.
- Web Services Description Language (WSDL) for a web services client application.
- Documentation in HTML form to give to programmers or business analysts.

These generation options are available via New menu actions, with the exception of message dictionary generation. This is only available if you are using WebSphere Business Integration Message Broker, and is automatically generated when you add your message set to a Broker Archive (BAR) file for deployment to a broker.

## Generate message dictionaries

A message dictionary is data structure which describes all of the messages in a message set in a form suitable for deployment to WebSphere Message Broker.

**Purpose of a message dictionary:** A dictionary describes the logical structure and content of a set of messages, and usually contains one or more physical formats which describe how those messages are serialized in a bitstream. The MRM parser within WebSphere Message Broker can use this information to parse an incoming message bitstream into a message tree, or to convert a message tree into a physical bitstream.

**Contents of a message dictionary:** A message dictionary contains the same information as the message set from which it was created, but in a compressed form which the MRM parsers within WebSphere Message Broker can understand and use. A message dictionary contains all the elements in the message set, the structure of the messages, and all the value constraints. A message dictionary also contains any physical formats which were defined in the message set.

**Generating a message dictionary:** Before a message dictionary can be used, it must be deployed to WebSphere Message Broker. This is achieved by adding the message set to a BAR file, and then deploying the BAR file to a message broker. The generation of the message dictionary is performed automatically when a message set is added to a BAR file.

Before adding a message set to a BAR file, the Message Brokers Toolkit performs a full validation of the message set. If this validation finds any errors, the message set is not added to the BAR file, and therefore no dictionary is generated.

**Dictionary generation report files:** Whenever a message dictionary is generated, a report file is also generated and added to the same BAR file. This report file contains informational messages and warnings which relate to the use of the generated dictionary. You should always check this file after generating a message dictionary.

## Generate XML Schema

XML Schema is a standard way of describing complex message models.

You can generate a schema file from a message definition file. When selecting the input message definition file, if any XML physical formats have been defined for the message set, you can select which of these XML wire formats are to be included.

- If an XML format has been selected, the physical format information will also be included.
- If no XML format is selected, the generated schema file only contains information about the logical message model.

You can choose whether 'strict' or 'lax' schema generation is to be performed. This is necessary because the logical extensions to the XML Schema model provided by the message definition file cannot be represented in XML Schema. So you can choose either to generate a Schema with more strict or more lax validation than the equivalent validation performed by the message model parser. The affected model extensions are:

- Content Validation = open
- Content Validation = open defined
- Composition = unordered set

**Further information about XML Schema:** For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

## WSDL

A Web Services Description Language (WSDL) document specifies the interface for a Web Service. A WSDL document enables a Web Services client to invoke a Web Service using the messages defined in a message definition.

You are recommended to use message definition files with target namespaces when you generate WSDL. If you do not do this, WebSphere Message Broker will default the target namespace to the WSDL target namespace.

Before running the WSDL generator, you should have created one or more message category files for your message set. The category file must have a Message Category Kind of wsdl. The category file should also have one or more messages and these must be assigned a role type of wsdl:input, wsdl:output or wsdl:fault. Each category corresponds to a WSDL Port Type in the generated message WSDL file.

For each message definition file in the message set, one XML schema file is generated. These files are included by the message WSDL file, and WSDL uses these to define its message types. The WSDL messages are then declared using these types. These are the same messages which were previously added into the category files.

You must select at least one binding for the output WSDL. The valid bindings are SOAP (over JMS), SOAP (over HTTP) and JMS (TextMessage).

WSDL bindings correspond to the physical format for the messages. A separate bindings WSDL file is generated as part of the output. All the selected bindings are described in this file, which refers to the message WSDL described above.

WSDL services describe the web service. These map the bindings described above onto a particular service. All the WSDL services for a message set will be defined in one separate file.

### **Summary of the files used by the WSDL generator:** Input Files:

- One or more Message definition files. They must exist in the message set
- One or more category files. They must exist within the message set and reference one or more messages in the message definition files.

### Output Files:

- One XML schema file. Generated for each message definition file in the message set.
- One WSDL message file. Generated for the message set. It references messages defined in the schema files.
- One WSDL bindings file. Generated for the message set. It contains all the possible bindings (up to three) for the messages.
- One WSDL services file. This defines the web service.

**Relationship to the message model for generating:** If a broker is to communicate with a web service client, it typically needs to accept SOAP messages. The recommended approach is to use the MRM domain, in which case the broker's message model and the WSDL definition used by the web service client must describe the same messages.

If the broker has an existing message model (possibly created by importing a C header file or COBOL copybook), this can be exported to create a corresponding WSDL definition for use by the client. At the same time, your message model needs to be enhanced with appropriate definitions for the SOAP envelope and (for rpc-style) the WSDL operations. Currently only WSDL version 1.1 is supported.

In order to generate WSDL you need:

1. a way of representing the WSDL operations. This is the role of the message category.
2. a way of representing the data for these operations. This is the message model.
3. a way of soliciting the web service end-point and binding details. This is the role of the WSDL Generator wizard.

A message category is required for each WSDL operation. The category specifies a set of messages from the broker model and associates them with the required WSDL qualifiers for the specified WSDL operation type.

At runtime, the format of the SOAP messages depends on the WSDL style specified in the wizard. If the user selects rpc-style then the SOAP Envelope will contain a message corresponding to a WSDL operation. The WSDL generator will then add an appropriate message definition that corresponds to the WSDL operation to your message set. If you select document-style then the SOAP envelope will simply contain messages specified in the category, so no additional message definitions need to be added to your message set.

Message definitions for the SOAP envelope and (if necessary) the SOAP encoding scheme are imported into the message set.

## Resulting message model

The resulting model allows you to parse incoming SOAP messages using the MRM XML parser where the message type would be Envelope. The message model for the SOAP envelope defines the outer SOAP wrapper with its constituent header and body sections and a number of attachment points where the various business payloads can appear. These attachment points are defined with composition of type message, allowing broker messages to appear at these points.

The allowed attachment points are Envelope.Body, Envelope.Header and Envelope.Body.Fault.detail. A message from your message model may appear at each point (in the case of the Envelope.Header, multiple messages may appear). In the case of rpc-style WSDL, the message expected at Envelope.Body is the automatically generated message corresponding to the WSDL operation (for example, the message category). In all other cases the messages expected are those referenced by the message categories.

## Generate documentation

Message set documentation describes, in a human-readable format, message definitions which you have created.

When you have created one or more message definitions, it can be useful to generate message set documentation for business analysis and for developers who are involved with the messages.

Message definition files contain both logical and physical definitions for the message model. The generated documentation describes the logical format only.

The documentation exists as a self-consistent set of HTML pages.

---

## Working with a message set project

Before you begin to develop your message model it is necessary to create a message set project.

This topic area describes the tasks that are involved in working with a message set project:

- “Creating a message set project”
- “Deleting a message set project” on page 80

### Creating a message set project

This task topic describes how to create a new message set project using the New Message Set Project wizard.

You must create a message set project before you can create a message set. You can optionally create a message set immediately after you create a message set project, or you can create the message set later.

**Tip:** You can only have one message set within a message set project. A message set project cannot refer to any other message sets.

To create a new message set project:

1. Switch to the Broker Application Development perspective.
2. Open the New Message Set Project wizard. To do this, right-click anywhere in the Resource Navigator then click **New > Message Set Project** on the pop-up menu.  
Alternatively, from the workbench menu, click **File > New > Message Set Project** or click the **Create a new message set project** icon on the toolbar at the top of the workbench window.
3. Type the **Project Name** for the message set project that you are creating.
4. Optional: At this point you can specify a directory in which you want to store the project contents. If you do not specify a directory the default is used. To specify a directory, first clear the **Use default** check box and then either type the path in **Directory** or click **Browse** to navigate to the folder that you want to use to store your project contents.
5. Optional: If you do not want to create a message set within the message set project at the moment, click **Finish**. Your message set project is created immediately and the New Message Set Project wizard automatically closes.
6. If you do now want to create a message set within the message set project, click **Next** to move to the next page of the wizard.
7. Type a **Message Set Name** then specify the properties for the message set as described in “Creating a message set” on page 81. Note that when you type the name, this enables the **Next** button, allowing you to move to the next page of the wizard.
8. Click **Finish**. A message set project is created with the name that you specified. A message set is also created within the message set project, with the name and properties that you specified.

## Deleting a message set project

### Before you start:

To complete this task, you must have completed the following task:

- “Creating a message set project” on page 79

**Tip:** It is recommended that you close all open windows within the workbench that relate to the message set project or associated files that you are deleting. Failure to do so can cause errors to occur in relation to the workbench attempting to process objects that no longer exist.

This task topic describes how to delete a message set project and, optionally, the contents of the associated project directory.

To delete a message set project:

1. Switch to the Broker Application Development perspective.
2. In the Resource Navigator, right-click the message set project that you want to delete, then click **Delete** on the pop-up menu. The Confirm Project Delete window opens.
3. Choose whether to delete or retain the contents of the project directory. By default, project directory contents are not deleted. If you choose to delete the contents of the project directory by clicking **Also delete contents**, all files and directories associated with the project are deleted.
4. Click **Yes** to delete the message set project. Alternatively, click **No** or press Esc to cancel the deletion.

---

## Working with a message set

### Before you start:

To complete the tasks in this topic area, you must have completed the following task:

- “Creating a message set project” on page 79

This topic area describes the tasks that are involved in working with a message set:

- “Configuring message set preferences”
- “Opening an existing message set” on page 81
- “Creating a message set” on page 81
- “Configuring logical properties: Message sets” on page 83
- “Working with physical properties” on page 84
- “Configuring documentation properties: Message sets” on page 89
- “Deleting a message set” on page 90
- “Applying a Quick Fix to a task list error” on page 90

## Configuring message set preferences

This task topic explains how to make changes to preferences relating to the handling of message set processing. These preferences are for message set editors, message set validation and importing XML schema.



To configure message set preferences:

1. Open the Preferences window by clicking **Window > Preferences**.
2. In the left hand pane, expand **Message Set** by clicking **+**. This displays the following options:
  - Editors
  - Validation
  - XML Schema Importer
3. View or make any necessary changes to the preferences for message set processing. These preferences are shown in the right hand area of the window.
4. When you have finished, click **Apply**. Alternatively, click **Restore Defaults** to return to the default settings for the displayed fields.
5. Close the Preferences window by clicking **OK**.

## Opening an existing message set

### Before you start:

To complete this task, you must have completed the following task:

- “Creating a message set”

**Tip:** Although you can open resource files with other editors you are advised to only use the workbench Message Set editor to work with message set files because this editor correctly validates changes made to the `messageSet.mset` files when they are saved. Other editors might not do this.

To open a message set so that you can view or edit its contents:

1. Switch to the Broker Application Development perspective.
2. In the Resource Navigator, right-click the `messageSet.mset` file of the message set that you are opening then click **Open** on the pop-up menu. This launches the Message Set editor for the selected file.

You can now view or edit the file as required.

## Creating a message set

### Before you start:

To complete this task, you must have completed the following task:

- “Creating a message set project” on page 79

This task topic describes how to create a message set within your message set project.

You must create a message set before you can add any content to your message model.

**Note:** A message set project can only contain one message set.

**Tip:** It is possible to create your message set at the same time as you create the message set project to which it belongs. For further information see “Creating a message set project” on page 79.

To create a new message set:

1. Switch to the Broker Application Development perspective.
2. Open the New Message Set wizard and select the message set project to which you want to add the message set. To do this right-click anywhere in the Resource Navigator then click **New > Message Set** on the pop-up menu. Alternatively, from the workbench menu, click **File > New > Message Set** or click the **Create a new message set** icon on the toolbar at the top of the workbench window.

In all cases, the New Message Set wizard opens with a list of message set projects that currently have no associated message set, from which you can select the message set project to which you want to add the message set.

**Tip:** If you have already decided which message set project you are adding the message set to, select this project in the Resource Navigator before you open the New Message Set wizard. As long as the message set project does not already have an associated message set, this project is then preselected when the wizard appears. (If your chosen message set project already has a message set associated with it, a list of message set projects without a message set is displayed instead.)

3. Type a name for your new message set in the **Message Set Name** field.
4. Optional: If you want to enable namespace support now, select the **Use namespaces** check box. Namespaces provide a method of avoiding naming conflicts where different document definitions have elements of the same name. Selecting to use namespaces allows the use of namespaces when you are defining message model objects. For further information see “Namespaces” on page 35.

**Tip:** If you enable namespace support at this point, you cannot then undo or remove this action at a later stage. However, you can add namespace support at a later stage if you do not add it now.

**Note:** Using namespaces also affects how elements are created in the WebSphere Message Broker logical message tree. Each element in the message tree will have both a name and a namespace, so an ESQL or Java reference to one of these elements has to specify both name and namespace. Therefore using namespaces has an effect on the ESQL or Java you write.

You should select this property if the message set will be used now or in the future to model XML messages. However, if the message set is to be used with message flows running on WebSphere Message Broker Version 2.1, or that originated on WebSphere Message Broker Version 2.1, you should *not* select this property, because such message flows will not be namespace aware.

5. Optional: If you want to base your new message set on an existing message set, in the **Base Message Set** list, click the message set that you want to use. If you choose to use a base message set, your new message set is an exact copy of the base message set but has a new unique message set ID and the name you specify. The base message you use remains unchanged and available for editing.
6. Optional: Click **Finish**. Your message set is created with the properties that you have just specified and the wizard closes.
7. If you want to add any physical formats to the message set before you create it, click **Next** to move to the next page of the wizard. To add a physical format,

select the appropriate check box then specify the name that you want to use for this physical format layer in the field to the right of the check box. You can either type a new name or accept the default name already shown for the selected physical format. The default names are as follows:

- **Custom Wire Format Name:** the default name is CWF1.
- **XML Wire Format Name:** the default name is XML1.
- **Tagged/Delimited Format:** the default name is TDS1.

The physical format or formats that you select are added to the message set when it is created.

#### 8. Click **Finish**.

Your message set is created and the new `messageSet.mset` file is automatically opened in the Message Set editor.

If you have chosen to add any physical format layers, these are also created with the name that you specified. They are displayed in the Properties Hierarchy in the Message Set editor.

A default package is created for objects that have not been qualified with a namespace.

## Configuring logical properties: Message sets

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81

If the `messageSet.mset` file for the appropriate message set is not already open in the Message Set editor, you must first open it as described in “Opening an existing message set” on page 81.

This task topic describes how to configure the logical properties of a message set using the Message Set editor.

To configure the logical properties of a message set:

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, in the Properties Hierarchy, click **Message Set**. This displays the logical properties of the selected message set in the Details view.
3. Configure the logical properties shown in the Details view to your requirements.

**Note:** Property fields that are disabled cannot be altered. For example, the **Message Set ID** field is disabled because the message set ID is generated automatically when the message set is created and must not then be altered.

4. Save your changes by clicking **File > Save messageSet.mset** or by pressing **Ctrl+S**. Alternatively click **File > Save All** or press **Ctrl+Shift+S**.

## Working with physical properties

### Before you start:

To complete the tasks in this topic area, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Opening an existing message set” on page 81

When you are developing your message model it is likely that you will want to add one or more physical format layers to a message set and then configure the properties of these physical formats. This topic area covers the following tasks relating to working with the physical properties of a message set:

- “Adding a Custom Wire Format (CWF)”
- “Configuring Custom Wire Format (CWF) properties: Message sets” on page 85
- “Adding an XML wire format” on page 85
- “Configuring XML Wire Format properties: Message sets” on page 86
- “Adding a Tagged/Delimited String (TDS) wire format” on page 86
- “Configuring Tagged/Delimited String (TDS) properties: Message sets” on page 87
- “Renaming a physical format” on page 88
- “Applying default physical format settings: Message sets” on page 88
- “Removing a physical format” on page 89

### Adding a Custom Wire Format (CWF)

#### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Opening an existing message set” on page 81

This task topic describes how to add a Custom Wire Format (CWF) physical format layer to a message set using the Message Set editor.

To add a CWF physical format layer to a message set:

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, in the Properties Hierarchy, open the Add Custom Wire Format window by right-clicking **Physical properties** and then clicking **Add Custom Wire Format** on the pop-up menu.
3. On the Add Custom Wire Format window, specify the name that you want to use for the new CWF physical format.

**Tip:** Physical format names must be unique across a message set. It is recommended that you start the name of your new CWF physical format with CWF as this clearly identifies the type of the physical format that you are adding.

4. Click **OK** to add the physical format layer to the message set. Alternatively, if you decide to cancel the process, click **Cancel** or press Esc on the keyboard.

**Tip:** The new physical format layer is added with the relevant default property values. You can configure the message set properties according to your requirements, using the appropriate editor.

5. Save your changes either by clicking **File > Save messageSet.mset** or by pressing Ctrl+S.

## Configuring Custom Wire Format (CWF) properties: Message sets

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Opening an existing message set” on page 81
- “Adding a Custom Wire Format (CWF)” on page 84

This task topic describes how to configure the Custom Wire Format (CWF) properties of a message set using the Message Set editor.

To configure the CWF properties of a message set:

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, the **Physical properties** node of the Properties Hierarchy shows the name of each of the physical formats that have been added to the message set. If the physical format names are not in view, expand **Physical properties** by clicking +.
3. Under **Physical properties**, click the CWF physical format so that the properties of this format appear in the Details view of the Message Set editor.

**Tip:** Physical formats can have any name but by default the CWF physical format is called CWF1.

4. Configure the CWF properties shown in the Details view according to your requirements.
5. Save your changes either by clicking **File > Save messageSet.mset** or by pressing Ctrl+S. Alternatively click **File > Save All** or press Ctrl+Shift+S.

## Adding an XML wire format

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Opening an existing message set” on page 81

This task topic describes how to add an XML wire format physical format layer to a message set using the Message Set editor.

To add an XML physical format layer to a message set:

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, in the Properties Hierarchy, open the Add XML Wire Format window by right-clicking **Physical properties** and then clicking **Add XML Wire Format** on the pop-up menu.

3. On the Add XML Wire Format window, specify the name that you want to use for the new XML wire format.

**Tip:** Physical format names must be unique across a message set. It is recommended that you start the name of your new XML physical format with XML as this clearly identifies the type of the physical format that you are adding.

4. Click **OK** to add the physical format layer. Alternatively, if you decide to cancel the process, click **Cancel** or press Esc on the keyboard.

**Tip:** The new physical format layer is added with the relevant default property values. You can configure the message set properties according to your requirements, using the appropriate editor.

5. Save your changes either by clicking **File > Save messageSet.mset** or by pressing Ctrl+S.

## Configuring XML Wire Format properties: Message sets

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Opening an existing message set” on page 81
- “Adding an XML wire format” on page 85

This task topic describes how to configure the XML Wire Format properties of a message set using the Message Set editor.

To configure the XML wire format properties of a message set:

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, the **Physical properties** node of the Properties Hierarchy shows the name of each of the physical formats that have been added to the message set. If the physical format names are not in view, expand **Physical properties** by clicking +.
3. Under **Physical properties**, click the XML wire format so that the properties of this format appear in the Details view of the Message Set editor.

**Tip:** Physical formats can have any name but by default the XML wire format is called XML1.

4. Configure the XML wire format properties shown in the Details view according to your requirements.
5. Save your changes either by clicking **File > Save messageSet.mset** or by pressing Ctrl+S. Alternatively click **File > Save All** or press Ctrl+Shift+S.

## Adding a Tagged/Delimited String (TDS) wire format

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Opening an existing message set” on page 81

This task topic describes how to add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

To add a TDS physical format layer to a message set:

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, in the Properties Hierarchy, open the Add Tagged/Delimited String Format window by right-clicking **Physical properties** then clicking **Add Tagged/Delimited String Format** on the pop-up menu.
3. In the Add Tagged/Delimited String Format window, specify the name that you want to use for the new TDS format.

**Tip:** Physical format names must be unique across a message set. It is recommended that you start the name of your new TDS physical format with TDS as this clearly identifies the type of the physical format that you are adding.

4. Click **OK** to add the physical format to the message set. Alternatively, if you decide to cancel the process, click **Cancel** or press Esc on the keyboard.

**Tip:** The new physical format layer is added with the relevant default property values. You can configure the message set properties according to your requirements, using the Message Set editor.

5. Save your changes either by clicking **File > Save messageSet.mset** or by pressing Ctrl+S.

## Configuring Tagged/Delimited String (TDS) properties: Message sets

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Opening an existing message set” on page 81
- “Adding a Tagged/Delimited String (TDS) wire format” on page 86

This task topic describes how to configure the Tagged/Delimited String (TDS) format properties of a message set using the Message Set editor.

To configure the TDS physical format properties of a message set:

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, the **Physical properties** node of the Properties Hierarchy shows the name of each of the physical formats that have been added to the message set. If the physical format names are not in view, expand **Physical properties** by clicking +.
3. Click the TDS format under **Physical properties** so that the properties of this format appear in the Details view of the Message Set editor.

**Tip:** Physical formats can have any name but by default the TDS physical format is called TDS1.

4. Configure the TDS properties shown in the Details view according to your requirements.
5. Save your changes either by clicking **File > Save messageSet.mset** or by pressing Ctrl+S. Alternatively click **File > Save All** or press Ctrl+Shift+S.

## Renaming a physical format

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Opening an existing message set” on page 81

This task assumes that you have added one or more physical formats to the message set that you are working with. For further information see “Adding a Custom Wire Format (CWF)” on page 84 or “Adding an XML wire format” on page 85 or “Adding a Tagged/Delimited String (TDS) wire format” on page 86.

This task topic describes how to rename a physical format using the Message Set editor.

To rename a physical format previously added to the message model:

1. Close all message definition (.mxsd) files that are currently open in the Message Definition editor, otherwise you will not be able to rename the physical format.
2. Switch to the Broker Application Development perspective.
3. In the Message Set editor, the **Physical properties** node of the Properties Hierarchy shows the name of each of the physical formats that have been added to the message set. If the physical format names are not in view, expand **Physical properties** by clicking +.
4. Under **Physical properties**, right-click the physical format that you want to rename then click **Rename** on the pop-up menu to open the “Rename wire format” window.
5. In the “Rename wire format” window, type the new name for the physical format. The renaming operation modifies all of the message definition files in the message set and saves the amended message set file.
6. Click **Finish** to rename the physical format and save the message set file.

## Applying default physical format settings: Message sets

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Opening an existing message set” on page 81

The tasks in this topic area assume that you have added one or more physical formats to the relevant message set. For further information see “Adding a Custom Wire Format (CWF)” on page 84 or “Adding an XML wire format” on page 85 or “Adding a Tagged/Delimited String (TDS) wire format” on page 86.

To apply the default settings to a physical format that has previously been added to a message set:

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, in the Properties Hierarchy, right-click the physical format to which you want to apply the default settings then click **Apply default physical format settings** on the pop-up menu.



The default settings are applied to the physical format that you have selected. No warning appears beforehand.

## Removing a physical format

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Opening an existing message set” on page 81

The tasks in this topic area assume that you have added one or more physical formats to a message set. For further information see “Adding a Custom Wire Format (CWF)” on page 84 or “Adding an XML wire format” on page 85 or “Adding a Tagged/Delimited String (TDS) wire format” on page 86.

To remove a physical format layer from your message set:

1. Close any message definition files that are currently open in the Message Definition editor, otherwise you will not be able to remove the physical format.
2. Switch to the Broker Application Development perspective.
3. In the Message Set editor, the **Physical properties** node of the Properties Hierarchy shows the name of each of the physical formats that have been added to the message set. If the physical format names are not in view, expand **Physical properties** by clicking +.
4. Under **Physical properties** right-click the physical format that you want to remove then click **Delete** on the pop-up menu.

**Tip:** If you decide to proceed with deleting the physical format, all of the message definition files under the current message set are modified and the amended message set file is saved.

5. Click **Finish** to remove the physical format, or click **Cancel** to return to the Broker Application Development perspective without making any changes. Alternatively, pressing Esc also returns you to the Broker Application Development perspective without making any changes.

## Configuring documentation properties: Message sets

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Opening an existing message set” on page 81

This task topic describes how to document a message set within the workbench.

To configure the documentation for a message set:

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor **Properties Hierarchy**, click **Documentation**. The documentation text field appears in the Details view.

3. Configure the documentation properties shown in the Details view to your requirements.

**Tip:** The Documentation property can be used to set user defined keywords and their value. These are propagated to the Configuration Manager when the message set is deployed to the broker. These keywords are used to give additional information about the message set when the Configuration Manager is used to display the message set properties.

4. Save your changes either by clicking **File > Save messageSet.mset** or by pressing Ctrl+S. Alternatively click **File > Save All** on the menu or press Ctrl+Shift+S.

## Deleting a message set

To delete a message set from your message model:

1. Switch to the Broker Application Development perspective.
2. In the Resource Navigator, right-click the message set folder that you are deleting, then click **Delete** on the pop-up menu. This opens the Confirm Resource Delete window, which asks whether you want to delete the resource that you have specified.
3. Click **Yes** to delete the message set or **No** to cancel the message set deletion. Pressing the Esc key on your keyboard also cancels the message set deletion.

**Important:** When you delete a message set, the action can not be undone once you have confirmed the deletion. All folders and associated files for the message set are deleted.

## Applying a Quick Fix to a task list error

This describes how to apply a quick fix to a task list warning or error.

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91

During the creation, migration and manipulation of message definition files, warnings or errors will occur and will be listed in the Problems view of the Broker Application Development perspective. A number of these warnings or errors can be cleared using Quick Fix. The types of warnings or errors that this will apply to are those where the construct has a broken link, where the construct has an illegal facet, or where the construct has been imported and where a warning or error has occurred but has been kept to ensure the integrity of structure that is being imported. This will allow you to fix the problem in the most appropriate way.

To apply a Quick Fix:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Problems view is visible in the Broker Application Development perspective of the workbench. If the Problems view is not visible, from the workbench menu, click **Window → Show View → Problems**.

3. In the Problems view, right-click the task list warning or error that you want to apply the Quick Fix to and then click **Quick Fix** on the pop-up menu. If Quick Fix is greyed out in the pop-up menu, then there is not one available for the problem you are trying to fix.
4. Step through the dialogue boxes that open and make any necessary selections to ensure that the fix is applied in the appropriate way.

When the Quick Fix has successfully been applied to the task list warning or error, it will be removed from the Problems view.

---

## Working with a message definition file

### Before you start:

To complete the tasks in this topic area, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81

This topic area describes the tasks that are involved in working with a message definition file:

- “Opening an existing message definition file”
- “Creating a message definition file”
- “Deleting a message definition file” on page 92

## Opening an existing message definition file

This task topic describes how to open an existing message definition file in the Message Definition editor so that you can view and edit its contents.

To open an existing message definition file:

1. Switch to the Broker Application Development perspective.
2. In the Resource Navigator, right-click the message definition file (file extension \*.msxd) that you want to open, then click **Open** on the pop-up menu. This launches the Message Definition editor for the message definition file that you have specified.

**Tip:** The Eclipse framework lets you open resource files with other editors. You are advised to only use the workbench Message Definition editor to work with message definition files because this editor correctly validates changes made to the message definition files. Other editors might not do this.

3. View or edit the data in the file as required.

## Creating a message definition file

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81

You must create a message definition file before you can create the message model objects. The message definition file contains the logical and physical model definitions of the objects in XML schema form.

To create an empty message definition file to contain your message model objects:

1. Switch to the Broker Application Development perspective.
2. Open the New Message Definition File wizard by clicking **File > New > Message Definition File** from the workbench menu. Alternatively, you can open the wizard by right-clicking the message set in the Resource Navigator that you are adding the message definition file to and clicking **New > Message Definition File** on the menu.
3. In the displayed list of options, click **Create a new message definition file** then click **Next**.
4. Step through the remainder of the wizard filling in the details as required.

The new empty message definition file with the name that you have specified and a file extension of \*.mxsd opens in the Message Definition Editor so that you can use the editor to create your own message definitions. If you have selected to use a target namespace, a directory structure based on the URI that you have supplied is created. The new message definition file is placed within this directory structure, which appears in the Resource Navigator, Outline view and Overview editor.

## Deleting a message definition file

To delete a message definition file from your message model:

1. Switch to the Broker Application Development perspective.
2. In the Resource Navigator, right-click the message definition file (file extension \*.mxsd) that you want to delete, then click **Delete** on the pop-up menu. Alternatively, select the message definition file that you want to delete in the Resource Navigator then from the menu bar, click **Edit > Delete**, or press the Delete key.
3. In the Confirm Resource Delete window, click **Yes** to delete the message definition file. Alternatively, click **No** or press the Esc key to cancel the message definition file deletion.

**Important:** All files and objects associated with the message definition file are deleted. This action cannot be undone.

---

## Working with message model objects

### Before you start:

To complete the tasks in this topic area, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91

This topic area describes the tasks that are involved in working with message model objects:

- “Adding message model objects” on page 93
- “Configuring message model objects” on page 105

- “Deleting objects” on page 117

## Adding message model objects

### Before you start:

To complete the tasks in this topic area, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91

Before starting any of the tasks in this topic area, you must first open the message definition file to which you want to add message model objects in the Message Definition editor. See “Opening an existing message definition file” on page 91 for further details.

This topic area describes the tasks that are involved in adding message model objects to a message definition file:

- “Adding a message”
- “Adding a message from a global element” on page 94
- “Adding a global element” on page 94
- “Adding a local element” on page 95
- “Adding an element reference” on page 95
- “Adding a wildcard element” on page 96
- “Adding a global attribute” on page 97
- “Adding a local attribute” on page 97
- “Adding an attribute reference” on page 98
- “Adding a wildcard attribute” on page 98
- “Adding a simple type” on page 99
- “Adding a complex type” on page 100
- “Adding a simple type to an element” on page 101
- “Adding a complex type to an element” on page 102
- “Adding a global group” on page 102
- “Adding an attribute group” on page 103
- “Adding a group reference” on page 104

### Adding a message

#### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

To add a message to your message definition file:

1. Switch to the Broker Application Development perspective.

2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click **Messages** then click **Add Message** on the pop-up menu. A simple message is immediately added to your message model and is assigned a default name.
4. Either type a new name for the message or press Enter to accept the default.

**Tip:** When you add a message to your message model, an associated complex type and global element with the same name as the message are also created. The global element and the message cannot have different names and changing the name of one changes the names of both. The complex type can be renamed.

You can now configure the properties of the message to your exact requirements. For further information on configuring message model objects see “Configuring message model objects” on page 105.

## Adding a message from a global element

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91
- “Adding a global element” (This must be a global element of complex type)

To add a message from a global element to your message model:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click **Messages** then click **Add Message From Global Element** on the pop-up menu to open the Add Message From Global Element window. This window lists all the global elements of a complex type that are not already associated with a message.
4. In the **Choose a Global Element** list, click the global element that you want to use to create your message.
5. Click **OK**. This immediately adds a message with the same name as the selected global element to your message model.

You can now configure the properties of the message to your exact requirements. For further information on configuring message model objects see “Configuring message model objects” on page 105.

## Adding a global element

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79

- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

To add a global element to your message model:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click **Elements and Attributes** then click **Add Global Element** on the pop-up menu. This adds a global element of type string to your message model, and assigns a default name.
4. Either type a new name for the global element or press Enter to accept the default.

You can now configure the global element to your requirements. For further information on configuring message model objects see “Configuring message model objects” on page 105.

## Adding a local element

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

This task assumes that you have previously added the relevant message, type, group or complex element to your message model.

To add a local element to a message, type, group or complex element:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click the object (message, type, group or complex element) to which you are adding a local element then click **Add Local Element** on the pop-up menu. A local element of type string is added to the message model and is assigned a default name.
4. Either type a new name for the local element or press Enter to accept the default.

You can now configure the local element to your exact requirements. For further information on configuring message model objects see “Configuring message model objects” on page 105.

## Adding an element reference

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

This task assumes that you have previously added the relevant message, type, global group or complex element to your message model.

To add an element reference to a message, type, global group or complex element:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click the object (message, type, group or complex element) to which you are adding the element reference, then click **Add Element Reference** on the pop-up menu. This adds a default element reference to the message model object that points to an existing global element. This existing global element may be in the current message definition file or in a message definition file defined under **Includes** or **Imports** for the current message definition file. For further information on Imports and Includes see “Linking from one message definition file to another” on page 118.

You can now configure the element reference to your exact requirements. For further information on configuring message model objects see “Configuring message model objects” on page 105.

## Adding a wildcard element

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

You can add a wildcard element to a message, type, group or complex element. This task assumes that you have previously added the relevant message, type, group or complex element to your message model.

To add a wildcard element to a message, type, group or complex element:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click the message model object (message, type, group or complex element) to which you are adding the wildcard element then click **Add Wildcard Element** on the pop-up menu. A wildcard element is added and appears in the Outline view.

You can now configure the wildcard element to your exact requirements. For further information on configuring message model objects see “Configuring message model objects” on page 105.



## Adding a global attribute

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

To add a global attribute to your message model:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click **Elements and attributes** then click **Add Global Attribute** on the pop-up menu. A global attribute of type string is immediately added and is assigned a default name.
4. Either type a new name for the global attribute or press Enter to accept the default.

You can now configure the global attribute to your requirements. For more information on configuring message model objects see “Configuring message model objects” on page 105.

## Adding a local attribute

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

You can add a local attribute to a message, complex type or complex element. This task assumes that you have previously added the relevant message, complex type or complex element to your message model.

To add a local attribute to a message, complex type or complex element:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click the object (message, complex type or complex element) to which you are adding the local attribute then click **Add Local Attribute** on the pop-up menu. A local attribute of type string is immediately added to the message model object and is assigned a default name.
4. Either type a new name for the local attribute or press Enter to accept the default.

You can now configure the local attribute to your requirements. For further information on configuring message model objects see “Configuring message model objects” on page 105.

## Adding an attribute reference

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

You can add an attribute reference to a message, complex type or complex element. This task assumes that you have previously added the relevant message, complex type or complex element to your message model.

To add an attribute reference to a message, complex type or complex element:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click the message model object (message, complex type or complex element) to which you are adding the attribute reference then click **Add Attribute Reference** on the pop-up menu. This adds a default attribute reference to the message model object that points to an existing global attribute.

You can now configure the attribute reference to your exact requirements. For further information on configuring message model objects see “Configuring message model objects” on page 105.

## Adding a wildcard attribute

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

You can add a wildcard attribute to a message, complex type or complex element. This task assumes that you have previously added the relevant message, complex type or complex element to your message model.

To add a wildcard attribute to a message, complex type or complex element:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.

3. In the Outline view, right-click the object (message, complex type or complex element) to which you are adding the wildcard attribute then click **Add Wildcard Attribute** on the pop-up menu. A wildcard attribute of type string is immediately added to the message model object and is assigned a default name.
4. Either type a new name for the wildcard attribute or press Enter to accept the default.

You can now configure the wildcard attribute to your requirements. For further information on configuring message model objects see “Configuring message model objects” on page 105.

## Adding a simple type

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

To add a simple type to your message model:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click **Types** then click either **Add Simple Type Restriction**, **Add Simple Type List**, or **Add Simple Type Union** on the pop-up menu.
  - For a restriction, a simple type of base type string is added, and assigned a default name.
  - For a list, a simple type of item type string is added, and assigned a default name.
  - For a union, a simple type with a single member type of string is added, and assigned a default name.
4. Either type a new name for the simple type or press Enter to accept the default.

You can now configure the simple type to your exact requirements.

If the simple type is a restriction:

- You can change the base type using the editor Properties view.
- You can set the value constraints associated with the simple type. See “Setting value constraints” on page 109.
- You can replace the base type with a new local simple type. In the Outline view right-click on the simple type and click one of:
  - **Add Simple Type Restriction**. This replaces the base type with a new simple type restriction, with a base type of string. You can configure the restriction as described in ‘If the simple type is a restriction’. The result is that the original simple type becomes a restriction of a restriction.
  - **Add Simple Type List**. This replaces the base type with a new simple type list, with an item type of string. You can configure the list as described in ‘If

the simple type is a list'. The result is that the original simple type becomes a restriction of a list. Note that this appears as a list in the editor, because a restriction of a list is itself a list, but you can also set certain value constraints.

If the simple type is a list:

- You can change the item type using the editor Properties view.
- You can replace the item type with a new local simple type. In the Outline view right-click on the simple type and click one of:
  - **Add Simple Type Restriction.** This replaces the item type with a new simple type restriction, with a base type of string. You can configure the restriction as described in 'If the simple type is a restriction'. The result is that the original simple type becomes a list of a restriction.
  - **Add Simple Type Union.** This replaces the item type with a new simple type union, with a single member type of string. You can configure the union as described in 'If the simple type is a union'. The result is that the original simple type becomes a list of a union.

If the simple type is a union:

- If the member type of string is not required, in the Outline view right-click on the string and click Delete.
- You can add further members to the union. In the Outline view right-click on the simple type and click one of:
  - **Add Union Member Type.** This adds a union member that is an existing simple type. Use the type selection dialog to select the simple type required.
  - **Add Local Member Type Restriction.** This adds a union member that is a new simple type restriction, with a base type of string. You can configure the restriction as described in 'If the simple type is a restriction' above.
  - **Add Local Member Type List.** This adds a union member that is a new simple type list, with an item type of string. You can configure the list as described in 'If the simple type is a list' above.
  - **Add Local Member Type Union.** This adds a union member that is a new simple type union, with a single member type of string. You can configure the new union as described in 'If the simple type is a union'.
- New members are added to the end of the union. To change the order of a member, in the Outline view select the member and drag it to the desired position within the union. Note that all union members that are existing simple types must occur ahead of all members that are local restrictions, lists and unions, so reordering is subject to this rule.

For further information on configuring message model objects see "Configuring message model objects" on page 105.

## Adding a complex type

### Before you start:

To complete this task, you must already have completed the following tasks:

- "Creating a message set project" on page 79
- "Creating a message set" on page 81
- "Creating a message definition file" on page 91
- "Opening an existing message definition file" on page 91

To add a complex type to your message model:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click **Types** then click **Add Complex Type** on the pop-up menu. A complex type is added and is assigned a default name.
4. Either type a new name for the complex type or press Enter to accept the default.

You can now configure the complex type to your requirements. For further information on configuring message model objects see “Configuring message model objects” on page 105.

## Adding a simple type to an element

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

This task assumes that you have previously added the relevant element to your message model.

To add a new simple type to an element:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, click the element to which you want to add a new simple type.
4. In the Message Definition editor, click the **Properties** tab.
5. In the Properties Hierarchy, click **Logical Properties > Global Element**.
6. In the Details view, in the **Type** property, select **(New Simple Type Restriction)**, **(New Simple Type List)**, or **(New Simple Type Union)** to open the relevant New Simple Type window to create a simple type of the type that you specify.
7. In the New Simple Type window, in the **Base Type** list, click the type that you want to use.
8. Optional: If you want to add the new simple type as a global simple type, select the **Create as Global Simple Type** check box and specify the name for your new simple type in the **Name** field.
9. Click **OK**. A simple type is immediately added to your message model.

Any changes that you make are reflected throughout where the element to which you have added a new simple type occurs.

## Adding a complex type to an element

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

This task assumes that you have previously added the element to which you are adding the complex type to your message model.

When you add a complex type to an element, you can either create a new complex type or derive a new complex type from an existing base type.

To add a new complex type to an element:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, click the element to which you want to add a new complex type.
4. In the Message Definition editor, click the **Properties** tab.
5. In the Properties Hierarchy, click **Logical Properties > Global Element**.
6. In the Details view, in the **Type** list, click (**New Complex Type**) to display the New Complex Type window.
7. If you want to create a new local complex type, click **Create a Local Complex Type** then, in the **Composition** list, click the option that you require.
8. If you want to derive a new local complex type from an existing base type:
  - a. Click **Derive a new Local Complex Type from existing base type**.
  - b. In the **Base Type** list, click the base type that you want to use. Depending on the base type you select, the **Composition** and **Derived By** lists might become active.
  - c. If the **Composition** and **Derived By** lists are active, click the options that you require in both lists.
9. If you want to add the new complex type as a global complex type, select the **Create as Global Complex Type** check box, and specify a name for your new complex type in the **Name** field.
10. Click **OK** to close the New Complex Type window and add the new complex type to your message model.

Any changes that you make are reflected throughout where the element to which you are adding the complex type occurs.

## Adding a global group

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79

- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

To add a global group to your message model:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click **Groups** then click **Add Group** on the pop-up menu. A global group is added to your message model and is assigned a default name.
4. Either type a new name for the global group or press Enter to accept the default.

You can now configure the global group to your requirements. For further information on configuring the properties of message model objects see “Configuring message model objects” on page 105.

## Adding a local group

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

You can add a local group to a message, type, global group or complex element. This task assumes that you have previously added the relevant message, type, global group or complex element to your message model.

To add a local group to a message, type, global group or complex element:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click the object (message, type, global group or complex element) to which you are adding the local group then click **Add Local Group** on the pop-up menu. A local group is immediately added to the message model with type composition set to sequence by default.

You can now configure the local group to your requirements. For further information on configuring message model objects see “Configuring message model objects” on page 105.

## Adding an attribute group

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79

- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

To add an attribute group to your message model:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click **Groups** then click **Add Attribute Group** on the pop-up menu. A global attribute group is added to your message model and is assigned a default name.
4. Either type a new name for the attribute group or press Enter to accept the default.

You can now configure the attribute group to your requirements. For further information on configuring the properties of message model objects see “Configuring message model objects” on page 105.

## Adding a group reference

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

You can add a group reference to a message, type, global group or complex element. This task assumes that you have previously added the relevant message, type, global group or complex element to your message model.

To add a group reference to a message, type, global group or complex element:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right click the object (message, type, global group or complex element) to which you want to add a group reference then click **Add Group Reference** on the pop-up menu. A group reference is immediately added to your message model.

You can now configure the group reference to your requirements. For further information on configuring the properties of message model objects see “Configuring message model objects” on page 105.

## Adding an attribute group reference

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79



- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

To add an attribute group reference to your message model:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click **Groups** then click **Add Attribute Group Reference** on the pop-up menu. An attribute group reference is added to your message model and is assigned a default name.
4. Either type a new name for the attribute group reference or press Enter to accept the default.

You can now configure the attribute group reference to your requirements using the Message Definition editor. For further information on configuring the properties of message model objects see “Configuring message model objects.”

## Configuring message model objects

### Before you start:

To complete the tasks in this topic area, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Adding message model objects” on page 93 (You must have added one or more objects to your message model)

Before starting any of the tasks in this topic area, you must first open the message definition file for which you want to configure message model objects in the Message Definition editor. See “Opening an existing message definition file” on page 91 for further details.

This topic area describes the tasks that are involved in configuring message model objects:

- “Renaming objects” on page 106
- “Reordering objects” on page 106
- “Copying objects” on page 107
- “Pasting objects” on page 107
- “Changing the type of an element” on page 108
- “Setting value constraints” on page 109
- “Configuring logical properties: Message model objects” on page 110
- “Configuring documentation properties: Message model objects” on page 111
- “Configuring physical properties” on page 112
  - “Configuring Custom Wire Format (CWF) properties: Message model objects” on page 113

- “Configuring XML Wire Format properties: Message model objects” on page 114
- “Configuring Tagged/Delimited String (TDS) properties: Message model objects” on page 115
- “Applying default physical format settings: Message model objects” on page 116
- “Deleting objects” on page 117

## Renaming objects

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91
- “Adding message model objects” on page 93 (You must have added one or more objects to your message model)

Objects in the workbench such as files, messages and elements can have different physical representations. Eclipse handles renaming differently depending on the object.

**Tip:** Not all objects can be renamed. For example, you cannot rename `messageSet.mset`.

Where an object can be renamed the usual way in which to do it is as follows:

1. Switch to the Broker Application Development perspective.
2. In the Outline view, right-click the object that you want to rename then click **Rename** on the pop-up menu. Alternatively, right-click the object in the Message Definition editor **Overview** tab then click **Rename** on the pop-up menu. In both cases, depending on the object, either a renaming dialog opens or you will now be able to edit the name of the object directly.
3. Type the new name for the object.
4. If the renaming dialog is open, either press Enter or click **OK**.

## Reordering objects

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91
- “Adding message model objects” on page 93 (You must have added one or more objects to your message model)

To reorder objects within a message definition file:

1. Switch to the Broker Application Development perspective.

2. Click the object that you want to move. For example, you could select a local element within a message in either the Outline view or Properties Hierarchy.
3. Use the mouse to drag the object to its new location.

**Tip:** As you drag the object and the mouse cursor passes between objects, a black line appears, showing where the current focus is. If you try to drag and drop the object to a location that it cannot be moved to (including objects that are highlighted as the cursor passes over them), the object remains in its original position when you release it.

## Copying objects

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91
- “Adding message model objects” on page 93 (You must have added one or more objects to your message model)

You can copy an object within a message definition file, such as a message for a local element object, or types for a complex type object. To copy an object:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective. If the Outline view is not visible, from the workbench menu click **Window > Show View > Outline**.
3. In the Outline view, right-click the message model object that you want to copy then click **Copy** on the pop-up menu. Alternatively, right-click the object in the Message Definition editor **Overview** tab then click **Copy** on the pop-up menu.

The object is now copied.

## Pasting objects

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91
- “Adding message model objects” on page 93 (You must have added one or more objects to your message model)
- “Copying objects”

You can paste objects that you have previously copied within the same message definition file, such as a message for a local element object, or types for a complex type object.

You can only copy and paste an object within the same message definition. You cannot copy an object and paste it into another message definition, either within the same message set or in a different message set.

To paste an object in the message definition from which you copied it:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click the location where you are going to paste the object then click **Paste** on the pop-up menu. Alternatively, right-click the object in the Message Definition editor **Overview** tab then click **Paste** on the pop-up menu. The object appears in the new location with a default name which you can change if you want to.
4. Either type a new name for the object or press Enter to accept the default.

**Note:** When you copy and paste objects within the message set, where physical properties exist for that object, these settings are not pasted, but are set to default values.

**Tip:** If you cannot select **Paste** from either menu, this indicates that you are attempting to paste the object into a location that is invalid. For example, this would happen if you tried to paste a complex type into a local element.

## Changing the type of an element

### Before you start:

To complete this task, you must already completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91
- “Adding message model objects” on page 93 (You must have added one or more objects to your message model)

You can change the type of an element in your message model to another existing type, or you can create a new simple type or a new complex type.

To change the type of an element to an existing type:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, click the element for which you want to change the type.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the bottom left corner of the editor area.
5. In the Properties Hierarchy select **Logical Properties > Global Element**. If **Global Element** is not in view, expand **Logical Properties** by clicking **+**.
6. In the Details view, in the **Type** property, click the new type you require.

**Tip:** If the type you require is not displayed, you can find it by clicking **(More...)** in the list. This displays the Type Selection window with additional options. If you know which type you require, specify the first letter in the text box at the top of the Type Selection window. Matching types are then displayed, making the selection process easier.

7. When you have selected the type you require, click **OK**.

The change to the element type is reflected throughout where the element occurs.

The task above explains how to switch to an existing type. If you want to create a new simple type or a new complex type, select **(New Simple Type Restriction)**, **(New Simple Type List)**, **(New Simple Type Union)**, or **(New Complex Type)** in the **Type** list (see step 6 above). For information on how to create a new simple type or a new complex type see “Adding a simple type to an element” on page 101 or “Adding a complex type to an element” on page 102.

## Setting value constraints

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91
- “Adding a simple type” on page 99 or “Adding a simple type to an element” on page 101 (You must have added one or more global or local simple types to your message model)

Value constraints are usually associated with a simple type; they refine a simple type by defining limits on the values which the simple type can represent. To set the value constraints associated with a simple type:

1. Switch to the Broker Application Development perspective.
2. In the Outline view, click the simple type you are updating. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. Display the **Properties** tab of the Message Definition Editor by clicking **Properties** in the bottom left corner of the editor area. The Properties Hierarchy displays the following nodes:
  - Logical Properties
  - Physical Properties
  - Documentation
4. In the Properties Hierarchy under **Logical Properties** click **Value Constraints**. This displays the current value constraints settings for the selected simple type in the Details pane.

**Tip:** If **Value Constraints** is not in view, expand **Logical Properties** by clicking **+**.

5. Set the value constraints for the selected simple type by making the appropriate changes to the information shown in the Details pane.

### Setting an enumeration:

An enumeration restricts which values can be set for the value constraint. For example, "ABC" and "123". Use this section to create a list of fixed values that the associated type must match.

To set an enumeration:

1. Click **Add** to the right of the **Enumerations** field. This adds an enumeration that has a default enumeration (for example enumeration1).
2. Type the data that you want to set for this value constraint.
3. Press Enter on your keyboard.
4. Repeat the above steps for each enumeration that you are adding.

#### Setting a pattern:

Set a pattern to indicate that the value constraint defines a string used as a regular expression that must be matched by the data in the associated type. The regular expression syntax supported is a subset of XML Schema regular expressions.

See *Regular expression syntax* in the 2.1 documentation for a list of supported regular expression syntax elements.

To set a pattern:

1. Select **Add** to the right of the **Patterns** field. This adds a pattern that has a default pattern (for example pattern1) and is in update mode.
2. Type the data that you want to set for this value constraint.
3. Press Enter on your keyboard.
4. Repeat the above steps for each pattern that you are adding.

## Configuring logical properties: Message model objects

### Before you start:

To complete this task, you must have completed the following tasks:

- "Creating a message set project" on page 79
- "Creating a message set" on page 81
- "Creating a message definition file" on page 91
- "Opening an existing message definition file" on page 91
- "Adding message model objects" on page 93 (You must have added one or more objects to your message model)

To configure the logical properties of an object that is part of the message model:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench and is displaying the following information:
  - The name of the message definition file
  - Messages
  - Types
  - Groups
  - Elements and Attributes

If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**. If the information listed above is not displayed, ensure

that the message definition file is open in the Message Definition editor. Message definition files have an `.mxsd` file extension.

3. In the Outline view, select the message model object for which you want to configure the logical properties:
  - a. Depending on the type of the object that you are selecting, expand **Messages, Types, Groups** or **Elements and Attributes** as appropriate by clicking **+**.
  - b. Click the object that you want to select within the expanded node.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the bottom left corner of the editor area. The Properties Hierarchy displays the following nodes:
  - Logical Properties
  - Physical Properties
  - DocumentationThe type (for example, Local Element or Global Element) of the message model object that you selected in the Outline view is displayed under each of these nodes.

If the items under **Logical Properties** are not in view, expand **Logical Properties** by clicking **+**.
5. Display the logical properties of the selected object in the Details view of the Message Definition editor, by clicking the appropriate item under **Logical Properties**.
6. Configure the logical properties of the selected item to your requirements by making the appropriate changes to the information shown in the Details view.
7. Save your changes by clicking **File > Save message\_definition\_file.mxsd** or by pressing **Ctrl+S**. Alternatively click **File > Save All** or press **Ctrl+Shift+S**.

## Configuring documentation properties: Message model objects

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91
- “Adding message model objects” on page 93 (You must have added one or more objects to your message model)

To configure the documentation properties of an object contained within a message definition file:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench and is displaying the following information:
  - The name of the message definition file
  - Messages
  - Types
  - Groups
  - Elements and Attributes

If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**. If the information listed above is not displayed, ensure that the message definition file is open in the Message Definition editor. Message definition files have an `.mxsd` file extension.

3. In the Outline view, select the message model object for which you want to configure the documentation properties by doing the following:
  - a. Depending on the type of the object that you are selecting, expand **Messages, Types, Groups** or **Elements and Attributes** as appropriate by clicking **+**.
  - b. Click the object you want to select within the expanded node.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the bottom left corner of the editor area. The Properties Hierarchy displays the following nodes:
  - Logical Properties
  - Physical Properties
  - Documentation

The type (for example, Local Element or Global Element) of the message model object that you selected in the Outline view is displayed under each of these nodes.

**Tip:** If the items under **Documentation** are not in view, expand **Documentation** by clicking **+**.

5. Display the logical properties of the selected object in the Details view by clicking the appropriate item under **Documentation**.
6. Configure the documentation properties of the selected item to your requirements by typing text into the **Documentation** text field. Right-clicking in the text field allows you to select options for undoing changes you have made, cutting or copying text from the text field, pasting text into the text field, deleting highlighted text or selecting all text in the field.
7. Save your changes by clicking **File > Save message\_definition\_file.mxsd** from the menu or pressing **Ctrl+S**. Alternatively, from the menu, click **File > Save All** or press **Ctrl+Shift+S**.

## Configuring physical properties

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91
- “Adding message model objects” on page 93 (You must have added one or more objects to your message model)

The tasks in this topic area assume that you have added one or more physical formats to a message set. For further information see “Adding a Custom Wire Format (CWF)” on page 84 or “Adding an XML wire format” on page 85 or “Adding a Tagged/Delimited String (TDS) wire format” on page 86.



When you have added objects to your message model it is likely that you will want to configure the physical properties of these objects. The following tasks relate to configuring the physical properties of message model objects:

- “Configuring Custom Wire Format (CWF) properties: Message model objects”
- “Configuring XML Wire Format properties: Message model objects” on page 114
- “Configuring Tagged/Delimited String (TDS) properties: Message model objects” on page 115
- “Applying default physical format settings: Message model objects” on page 116

### **Configuring Custom Wire Format (CWF) properties: Message model objects:**

#### **Before you start:**

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91
- “Adding a Custom Wire Format (CWF)” on page 84
- “Adding message model objects” on page 93 (You must have added one or more objects to your message model)

To configure the CWF properties of a message model object:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench and is displaying the following information:
  - The name of the message definition file
  - Messages
  - Types
  - Groups
  - Elements and Attributes

If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**. If the above hierarchy is not displayed, ensure that the message definition file is open in the Message Definition editor. Message definition files have an `.mxsd` file extension.

3. In the Outline view, select the object for which you want to configure the CWF properties by doing the following.
  - a. Depending on the type of the object that you are selecting, expand **Messages, Types, Groups** or **Elements and Attributes** by clicking **+**.
  - b. Click the object you that want to select within the expanded node.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the bottom left corner of the editor area. In the Message Definition editor, in the Properties Hierarchy, the name of each of the physical formats that have been added to the message set appears under **Physical Properties**. The object type (for example, Local Element or Global Element) of the message model object that you selected in the Outline view is displayed under each physical format shown.

**Tip:** If the physical formats are not in view in the Properties Hierarchy, expand **Physical Properties** by clicking **+**. By default the CWF physical format is called CWF1 but could have a user defined name instead.

5. Under **Physical Properties**, click the object type for the message model object that you have chosen to configure under the CWF physical format. The CWF properties of your selected message model object appear in the Details view.
6. Configure the CWF properties of the selected object to your requirements by making the appropriate changes to the information shown in the Details view.

**Note:** It is not possible to configure disabled fields.

7. Save your changes by clicking **File > Save message\_definition\_file.mxsd** or pressing Ctrl+S. Alternatively click **File > Save All** or press Ctrl+Shift+S.

### Configuring XML Wire Format properties: Message model objects:

#### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91
- “Adding an XML wire format” on page 85
- “Adding message model objects” on page 93 (You must have added one or more objects to your message model)

To configure the XML Wire Format properties of a message model object:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench and is displaying the following information:
  - The name of the message definition file
  - Messages
  - Types
  - Groups
  - Elements and Attributes

If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**. If the above hierarchy is not displayed, ensure that the message definition file is open in the Message Definition Editor. Message definition files have an `.mxsd` file extension.

3. In the Outline view, select the object for which you want to configure the XML Wire Format properties by doing the following:
  - a. Depending on the type of the object that you are selecting, expand **Messages, Types, Groups** or **Elements and Attributes** by clicking **+**.
  - b. Click the object that you want to select within the expanded node.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the bottom left corner of the editor area. In the Message Definition editor, in the Properties Hierarchy, the name of each of the physical formats that have been added to the message set appears under **Physical Properties**. The object type (for example, Local Element or Global Element) of the message model object that you selected in the Outline view is displayed under each physical format shown.

**Tip:** If the physical formats are not in view in the Properties Hierarchy, expand **Physical Properties** by clicking **+**. By default the XML Wire Format is called XML1 but could have a user defined name instead.

5. Under **Physical Properties**, under the XML Wire Format, click the object type for the message model object that you have chosen to configure. The XML Wire Format properties of your selected message model object appear in the Details view of the Message Definition editor.
6. Configure the XML Wire Format properties of the selected object to your requirements by making the appropriate changes to the information shown in the Details view.

**Note:** It is not possible to configure disabled fields.

7. Save your changes by clicking **File > Save message\_definition\_file.mxsd** or pressing Ctrl+S. Alternatively select **File > Save All** from the menu or press Ctrl+Shift+S.

### Configuring Tagged/Delimited String (TDS) properties: Message model objects:

#### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91
- “Adding a Tagged/Delimited String (TDS) wire format” on page 86
- “Adding message model objects” on page 93 (Adding one or more objects to your message model)

To configure the TDS properties of a message model object:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench and is displaying the following information:
  - The name of the message definition file
  - Messages
  - Types
  - Groups
  - Elements and Attributes

If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**. If the above hierarchy is not displayed, ensure that the message definition file is open in the Message Definition editor. Message definition files have an `.mxsd` file extension.

3. In the Outline view, select the object for which you want to configure the TDS properties by doing the following:
  - a. Depending on the type of the object that you are selecting, expand **Messages**, **Types**, **Groups** or **Elements and Attributes** by clicking **+**.
  - b. Click the object that you want to select within the expanded node.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the bottom left corner of the editor area. In the Message Definition editor, in the Properties Hierarchy, the name of each of the physical formats that have been added to the message set appears under **Physical**

**Properties.** The object type (for example, Local Element or Global Element) of the message model object that you selected in the Outline view is displayed under each physical format shown.

**Tip:** If the physical formats are not in view in the Properties Hierarchy, expand **Physical Properties** by clicking **+**. By default the TDS physical format is called TDS1 but could have a user defined name instead.

5. Select the **Properties** tab located in the bottom left corner of the Message Definition editor.
6. Under **Physical Properties**, under the TDS physical format, click the object type for the message model object that you have chosen to configure. The TDS physical format properties of your selected message model object appear in the Details view.
7. Configure the TDS physical format properties of the selected object to your requirements by making the appropriate changes to the information shown in the Details view.

**Note:** It is not possible to configure disabled fields.

8. Save your changes by selecting **File > Save message\_definition\_file.mxsd** from the menu or press Ctrl+S. Alternatively, from the menu, select **File > Save All**, or press Ctrl+Shift+S.

#### **Applying default physical format settings: Message model objects:**

##### **Before you start:**

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91
- “Adding message model objects” on page 93 (You must have added one or more objects to your message model)

This task assumes that you have added one or more physical formats to the relevant message set. For further information see “Adding a Custom Wire Format (CWF)” on page 84 or “Adding an XML wire format” on page 85 or “Adding a Tagged/Delimited String (TDS) wire format” on page 86.

To apply the default physical format setting to a message model object previously added to a message definition file:

1. Switch to the Broker Application Development perspective.
2. In the Outline view, click the object to which you want to apply default physical format settings.
3. Click the **Properties** tab located in the bottom left corner of the Message Definition editor.
4. Check that the Message Definition editor Properties Hierarchy displays the following information:
  - Logical Properties
  - Physical Properties (For each of the physical formats that have been added to the message set, the name of the physical format appears under **Physical**

**Properties.** Under each physical format the type of message model object that you selected is displayed as a child.)

- Documentation

Ensure that **Physical Properties** in the Properties Hierarchy is fully expanded by clicking +.

5. Right click on the message model object type underneath the physical format to which you want to apply the default settings then click **Apply default physical format settings**. The default physical format settings for the message model object that you selected are applied without warning.
6. Save your changes by clicking **File > Save message\_definition\_file.mxsd** from the menu or pressing Ctrl+S. Alternatively, from the menu, click **File > Save All**, or press Ctrl+Shift+S.

## Deleting objects

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91
- “Adding message model objects” on page 93 (You must have added one or more objects to your message model)

To remove objects contained within a message definition file:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, right-click the object that you wish to remove then click **Delete** on the pop-up menu. Alternatively right-click the object in the Message Definition editor **Overview** tab then click **Delete** on the pop-up menu.

The type of object and the relationship that it has with other objects determines whether the object is now deleted without a confirmation window appearing, or whether a confirmation window opens with a list of all the objects that will be deleted along with the one that you have selected.

4. If a confirmation window opens, click **OK** to delete the objects.

**Tip:** You can undo a deletion by selecting **Edit > Undo**, as long as you have not saved the changes that you have made.

---

## Creating a multipart message

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91

- “Opening an existing message definition file” on page 91

A multipart message occurs when you embed a message within another message. To create a multipart (embedded) message:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, add one of the following objects to your message model:
  - A complex type (for further information on completing this task see “Adding a complex type” on page 100)
  - A global group (for further information on completing this task see “Adding a global group” on page 102)
  - A local group (for further information on completing this task see “Adding a local group” on page 103)

**Tip:** You can also use a local ANONYMOUS complex type when creating a multipart message. For further information see “Adding a complex type to an element” on page 102.

4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the bottom left corner of the editor area.
5. In the Properties Hierarchy, under **Logical properties**, click one of the following, depending on which of these you added in step 3:
  - **Complex Type**
  - **Global Group**
  - **Local Group**
6. In the Details view, make the following changes to the displayed logical properties:
  - a. In the **Composition** drop-down list, click **message**.
  - b. In the **Content validation** drop-down list, click **Open**, **Closed** or **Open Defined**, depending on your requirements. Note that if the embedded message is defined in a different message set, then you must click **Open**. For further information about using these three options see “Content Validation properties for complex types” on page 182.

**Note:** There are a number of different ways for the parser to identify an embedded message within a message bit stream. For further information on identifying a message within another message see the concept topics listed below.

---

## Linking from one message definition file to another

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Opening an existing message definition file” on page 91

There are two possible ways to link one message definition file to another: you can either add an include or you can add an import for the file that you want to reference.

When you are working with a message definition file, you can check which other message definition files this file currently includes or imports by carrying out the steps listed below:

1. Open the message definition file in the Message Definition editor.
2. In the Outline view, in the displayed hierarchy, select the .mxsd file.
3. In the Properties Hierarchy, expand **Imports** or **Includes** as appropriate to display a list of the other files that the currently selected file includes or imports.

## Include

You use the include option if you want to link to a message definition file with the same namespace, or if you want to link a message definition file with no target namespace to a message definition file with a target namespace (chameleon behavior). You must also choose to add an include rather than an import if you want to link a message definition file with no target namespace to another message definition file that also has no target namespace.

**Note:** A message definition file can only reference objects in another message definition file if this other file has been included directly, so you might have a problem if you try to use the include option to include message definition files that are themselves included within other message definition files. For information about ways of resolving this situation, see Resolving problems when developing message models.

This task assumes that you have opened an existing message definition file.

To add an include to a message definition file:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, click the message definition (.mxsd) file name.
4. Display the **Properties** tab of the Message Definition Editor by clicking **Properties** in the bottom left corner of the editor area.
5. In the Properties Hierarchy, right-click **Includes** then click **Add Include** on the pop-up menu. The “Select Message Definition file to include” window opens.
6. In the Message Sets pane, select the message definition file that you want to include. If the message definition files within your project are not visible in this pane, expand the project hierarchy by clicking **+**.
7. Click **Finish**. The message definition file that you selected in step 4 is included within the message definition file that you opened before beginning this task.

## Import

You use the import option if you want to link a message definition file to another message definition file in a different namespace. You cannot add an import from the same namespace; this includes linking from a message definition file with no target namespace to another message definition file with no target namespace.

To add an import to a message definition file:

1. Switch to the Broker Application Development perspective.
2. Ensure that the Outline view is visible in the Broker Application Development perspective of the workbench. If the Outline view is not visible, from the workbench menu, click **Window > Show View > Outline**.
3. In the Outline view, click the message definition (.mxsd) file name.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the bottom left corner of the editor area.
5. In the Properties Hierarchy, right-click **Imports** then click **Add Import** on the pop-up menu. The “Select Message Definition file to import” window opens.
6. In the Message Sets pane, select the message definition file that you want to import from the workspace. If the message definition files within your project are not visible in this pane, expand the project hierarchy by clicking **+**.
7. Click **Finish**. The message definition file that you selected in step 4 is imported into the schema of the message definition file that you opened before beginning this task.

---

## Working with a message category file

This topic area lists the tasks that are involved when working with a message category file.

- “Creating a message category file”
- “Opening an existing message category file” on page 121
- “Adding a message to a message category” on page 121
- “Deleting a message from a message category” on page 122
- “Viewing or configuring message category file properties” on page 123
- “Deleting a message category file” on page 123

## Creating a message category file

This topic describes how you would create a message category file.

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81

To create a new message category file:

1. Switch to the Broker Application Development perspective.
2. Open the New Message Category File wizard by right-clicking in the Resource Navigator then clicking **New > Message Category File** on the pop-up menu.

**Tip:** To preselect the message set when the wizard opens, either right-click the message set to which you are adding the message category file, or select the message set, before opening the wizard as just described.

3. In the first pane, select the Category Kind for the type of category that you are creating.
  - **other.** Indicates that this Message Category represents a generic grouping of messages. The Category Usage field is disabled.



- wsdl. Indicates that this Message Category represents a WSDL operation. The specified Category Name will become the WSDL operation name.
4. If the Category Kind has been set to wsdl, specify the WSDL operation type by selecting one of the following values for the Category Usage field:
    - wsdl:request-response
    - wsdl:solicit-response
    - wsdl:one-way
    - wsdl:notification
  5. Click **Next**. In the **Message Set Folder** field, select a folder under the target message set for the new message category file to be saved.
  6. In the **File name** field, type a name for the new message category file (It will automatically be given the file extension of .category).
  7. Click **Next**. Select any messages that you want to add to the new category. Use **Shift-click** to select a range of messages and **Ctrl-click** to select or deselect individual messages. You will not be able to complete the creation of the category file without adding one or more messages.
  8. Click **Finish**. A message category file is created within the message set folder that you selected, with the name that you specified and a file extension of .category.

The new message category file opens in the Message Category editor so that you can view and edit it as required. If the Category Kind was set to wsdl, then the messages that were added to the category will have Role Type set to wsdl:input and Role Usage set to soap:body.

## Opening an existing message category file

This describes how to open an existing message category file in the Message Category editor so that you can view or edit it.

### Before you start:

To complete this task, you must have completed the following task:

- “Creating a message category file” on page 120

To open an existing message category file:

1. Switch to the Broker Application Development perspective.
2. In the Resource Navigator, right-click the message category file (with a file extension of .category) that you want to open, then click **Open** on the pop-up menu. This opens the message category file that you have selected in the Message Category editor.
3. View and edit the message category file as required.

**Tip:** The Eclipse framework lets you open resource files with other editors. You are advised to only use the workbench Message Category editor to work with the message category files because this editor correctly validates changes made to the files. Other editors might not do this.

## Adding a message to a message category

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message category file” on page 120
- “Opening an existing message category file” on page 121

**Important:** This topic assumes that you have already added one or more messages to your message model.

To add a message to a message category file:

1. Switch to the Message Category editor, in the Broker Application Development perspective.
2. In the Properties Hierarchy, open the Add Messages window by right-clicking **Message Category** then clicking **Add Messages** on the pop-up menu. The Add Messages window lists all the messages that are available for adding to the message category file. Any message that is in the message set but has not already been added to the category is displayed.
3. Select the message or messages that you would like to add. Use **Shift-click** to select a range of messages and **Ctrl-click** to select or deselect individual messages.
4. Click **OK**. The selected message or messages are added to the message category and now appear in the Properties Hierarchy.

**Tip:** Until you save the message category file, you can undo any additions that you make. To undo a change, right-click **Message Category** in the Properties Hierarchy then click **Undo** on the pop-up menu. If you have added multiple messages, this removes all the messages that you have added. If you want to remove a single message, right click this message then click **Undo**. To redo an addition after undoing it, use the **Redo** option.

5. Save and validate the additions that you have made to the message category file by clicking **File > Save** or pressing Ctrl+S.

**Note:** Once you save the message category file after adding a message, you can no longer undo the addition of this message using the **Undo** option. If you want to remove a message after saving your changes, you can do so by deleting the message from the message category file.

Once you have added a message to a message category file, you can configure its properties according to your requirements in the Message Category editor Details view.

## Deleting a message from a message category

**Before you start:**

To complete this task, you must have completed the following tasks:

- “Creating a message category file” on page 120
- “Opening an existing message category file” on page 121
- “Adding a message to a message category” on page 121

To delete a message from a message category file:

1. Switch to the Broker Application Development perspective.
2. In the Message Category editor, in the Properties Hierarchy, right-click the message that you want to delete, then click **Delete** on the pop-up menu.

**Tip:** The message is deleted from the message category file immediately, without a warning appearing first.

## Viewing or configuring message category file properties

This topic describes how to view or configure the properties of a message category file and associated messages using the Message Category editor.

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message category file” on page 120
- “Opening an existing message category file” on page 121
- “Adding a message to a message category” on page 121 (You must have added one or more messages to your message category file)

To configure the properties of a message category file:

1. Switch to the Message Category editor in the Broker Application Development perspective.
2. To view or configure the properties of a message category, click **Message Category** in the Properties Hierarchy. From the Details section of the Message Category editor you can now view the properties of the message category and make any changes to the properties that are necessary.
3. To view or configure the properties of a message in the message category file, click the name of the message in the Properties Hierarchy. From the Details section of the Message Category editor you can now view the properties of the message and make any changes to the properties that are necessary.
4. If you have changed any of the properties in the message category or messages, you can save those changes by selecting **File** → **Save** from the menu.

**Note:** Note that some combinations of Message Category Usage, Role Type and Role Usage are not valid for WSDL and will result in task list errors being generated.

## Deleting a message category file

### Before you start:

To complete this task, you must have completed the following task:

- “Creating a message category file” on page 120

To delete a message category file:

1. Switch to the Broker Application Development perspective.
2. In the Resource Navigator, right-click the message category file (\*.category file extension) that you want to delete, then click **Delete** on the pop-up menu. Alternatively select the message category file in the Resource Navigator and then either click **Edit** > **Delete**, or press the Delete key on your keyboard.
3. On the Confirm Resource Delete window, click **Yes** to delete the message category file. Alternatively, to cancel the message category file deletion, either click **No** or press the Esc key.

**Tip:** Once you have deleted a message category file, the action cannot be undone.

---

## Importing

This describes how to create a message definition file by importing from a number of different data structures.

You can create a message definition file in a message set by importing from XML Schema, XML DTD, IBM supplied messages, WSDL definitions, C header files, and COBOL copybooks. This topic area describes how to import from these data structures using the command line or the workbench.

Before you attempt to create a message definition from a data structure, using the workbench, you are advised to read "Importing file systems into the workbench."

The following tasks topics relate to importing using the workbench:

- "Importing from C" on page 126
- "Importing from COBOL copybooks" on page 128
- "Importing from IBM supplied messages" on page 131
- "Importing from WSDL" on page 131
- "Importing from XML DTD" on page 134
- "Importing from XML schema" on page 136

The following tasks relate to importing using the command line:

- "Importing from the command line" on page 127 for C header files, COBOL Copybooks, XML DTDs and XML Schemas.
- "Importing WSDL definitions from the command line" on page 133

## Importing file systems into the workbench

### Before you start:

To complete this task, you must have completed the following task:

- "Creating a message set project" on page 79 (You only need to create a project. It is not necessary to create a message set.)

Before the workbench can use files to create a message definition based on a WSDL definition, an XML schema, XML DTD, C header file, or COBOL copybook, the files must be imported or copied into its file structure. This topic explains the three ways in which you can do this:

- Using the Import wizard to import files from a specified source.
- Dragging and dropping files from a file system.
- Copying and pasting files from a file system.

Using any of these import methods ensures that an imported file is available for use by your selected message set project. This means that you can select the imported file in the New Message Definition File wizard when you are creating a message definition based on this file.

### Using the Import wizard

You use the Import wizard to import all the files or a selection of files from the specified source.

To import files using the Import wizard:

1. Switch to the Broker Application Development perspective.
2. In the Resource Navigator click the project folder that you are going to import to.
3. Open the Import wizard by clicking **File > Import** on the workbench menu.
4. On the Select page of the Import wizard, click either **File System** or **Zip file**, depending on the type of resource that you are importing.
5. Click **Next**.
6. On the File System page, in the **Directory** field, specify the import source. To do this, type the source name in the field. Alternatively, click **Browse**, select the parent directory, or zip file, containing the file or files to import then click **OK** (directory) or **Open** (zip file).

**Tip:** Recent directories that you have imported from are shown in the drop-down list in the **Directory** field.

7. Using the left and right panes that appear under the **Directory** field, specify which folders or files, or both, you want to import. Note the following points when you are making your selections:
  - To import the entire contents of a folder into the workbench, select the check box for this folder in the left pane. To view any secondary folders within a folder, expand the folder by clicking +.
  - To import a specific file or files within a folder, use the right pane to select the individual files that you want to import. If you select a file or files in the right pane, the check box for the folder containing these files is greyed in the left pane to indicate that only some of the files in the folder will be imported into the workbench.
  - To restrict the type of files that you are going to import, click **Filter Types...** then, on the Select Types window, select the check boxes for the file types you want to include and click **OK**. If you want to include files with extensions that are not shown in the list, type these extensions in the **Other Extensions** field.
  - To select all the folders and files shown on the File System page, click **Select All**.
  - To deselect all folders and files that are currently selected on the File System page, click **Deselect All**.

**Note:** The **Select the destination for imported resources:** field should already be filled in with the name of the project folder you selected in step 2.

8. Optional: To change the destination project or folder click **Browse....** This opens the Folder Selection window. Select an alternate project folder to import into by clicking the folder then clicking **OK**.
9. Optional: To overwrite existing resources without a warning being displayed select the **Overwrite existing resources without warning** check box. This applies to both zip files and file systems.
10. File system import only: Select one of the following, depending on the folder structure that you want to create:
  - **Create complete folder structure**
  - **Create selected folders only**
11. Click **Finish**.

The files that you selected are imported and are now shown in the Resource Navigator under the project folder that you selected.

## Dragging and dropping

You can use dragging and dropping as a method to import files from your file system into the workbench. You must drag the resource(s) that you are importing to the exact location in the Resource Navigator where you want the resources to reside. You cannot simply drag and drop them onto a blank area in the Resource Navigator.

To import files by dragging and dropping:

1. In your file system, locate the file or folder that you want to import into the workbench.
2. Drag the file or folder to a specific location in the Resource Navigator. When you are dragging resources into the Resource Navigator, the project or folder that you are trying to drop the resource into is selected.
3. Ensure that the file or folder is copied into the workbench.

## Copying and pasting

You can use your operating system's copy and paste function as a method of importing a file system into the workbench.

To import files by copying and pasting:

1. Locate the file or directory that you want to import into the workbench.
2. Using your operating system's copy and paste function, copy the file or directory to your system's clipboard.
3. Select the destination for the file or directory in the Resource Navigator.
4. From the workbench menu, click **Edit > Paste**.

The files or directories are copied into the workbench and placed into the location that you selected.

## Importing from C

This describes how to create a new message definition from a C header file using the New Message Definition File wizard in the workbench.

### Before you start:

To complete this task, you must have completed the following tasks:

- "Creating a message set project" on page 79
- "Creating a message set" on page 81
- "Importing file systems into the workbench" on page 124

Before you begin this task, you should be aware of the points listed below:

- To create a new message definition file from a C header file, the header file must already be present in the workbench, for example in your message set project. This allows you to select the header file in the New Message Definition File wizard.
- The wizard allows you to import C header files with `.h`, `.c` and `.css` extensions. If your source file has a different extension you must rename it before attempting to import it.
- If the message set to which you are adding the new message definition file does not have an Custom Wire Format (CWF) layer only the logical information

appears in the model. You can add the physical layer to the message set before or after importing a C header file but it is recommended that you add the physical layer before you import the file.

- It is also possible to import a C header file from the command line using `mqsicreatemsgdefs`.

If you previously used the WebSphere MQ Integrator Version 2.1 C importer to create a message model, you may find that the WebSphere Message Broker Version 6.0 importer does not produce exactly the same results. In particular, the Version 2.1 C importer did not support as many C constructs as WebSphere Message Broker Version 6.0, and the naming convention for Version 2.1 compound types is not the same as the naming convention for WebSphere Message Broker Version 6.0 complex types and groups. In a very few cases, the naming convention for Version 2.1 elements is not the same as the naming convention for WebSphere Message Broker Version 6.0 elements.

The following steps cover both creating a completely new message definition file and overwriting the contents of an existing file.

To create a message definition file from a C header file:

1. Switch to the Broker Application Development perspective.
2. Open the New Message Definition File wizard by clicking **File > New > Message Definition File** from the workbench menu. Alternatively, you can open the wizard by right-clicking a C header file previously imported into the workbench and clicking **New > Message Definition File** on the menu.
3. In the displayed list of options, click **C header file** then click **Next**.
4. Step through the remainder of the wizard filling in the details as required.

When you have completed importing the C header file using the wizard:

- Carefully check for any errors in the report that is created when the file is imported. You can find this report in the log directory within the project containing the message definition that you have attempted to create. The report has a `.c.report.txt` file extension, prefixed with the name that you specified for the new message definition file.
- Review the messages shown in the workbench task list to check whether any new warnings or errors have appeared.

## Importing from the command line

This describes how to use the command line importer `mqsicreatemsgdefs` to import C, COBOL copybooks, XML DTD or XML schema in order to populate a message set with message definitions.

### Before you start:

Before you attempt the task described in this topic, it is recommended that you read the following reference topic:

- `mqsicreatemsgdefs` command

The command line importer allows you to create a new message set, into which the message definition files will be placed. When you create a new message set from the command line, only the logical information is created by default. However, the command line importer allows you to create a new message set based on an existing message set. The physical format information from the base

message set is also created in the new message set. If you want physical format information to be created as well, you must do the following before you invoke the **mqscreatmsgdefs** command:

1. Using the workbench, create a message set in your workspace that is to be used as a base message set.
2. To this base message set, add the physical formats that you want to be created in your new message set.

To import C, COBOL copybooks, XML DTD or XML schema using the command line:

1. Close the workbench. This must not be running when you use the command line importer.
2. Invoke the **mqscreatmsgdefs** command from a command prompt specifying the message set project name, path name of the source files folder, and any other optional parameters that you require. If you want to add physical formats to the new message set that the **mqscreatmsgdefs** command creates, specify the base message set that contains these physical formats as the **-base** parameter on the import command line.
3. When the command has completed, open `mqscreatmsgdefs.report.txt`. This report is created when you invoke the **mqscreatmsgdefs** command and by default is written to the directory from which you invoked the command. The report provides you with the following information:
  - Details of the parameters that were used when **mqscreatmsgdefs** was invoked.
  - The message set level action.
  - The name of the file or files that have been imported.
  - Details of the import process (for example, any warnings that have been generated and message model objects that have been created).
  - The number of files imported.
4. Start the workbench and switch to the Broker Application Development perspective. The message definition file that was created when you invoked **mqscreatmsgdefs** is visible in the project that you specified.

If an error occurs during the import of a C, COBOL copybook, XML DTD, or XML schema file, carefully check any errors that the importer reports. By default, all errors are written to the screen and to the log file described above. To gather additional information about the import, specify the **-v** (Verbose) command line parameter. This parameter displays more detailed information as the import proceeds.

## Importing from COBOL copybooks

This describes how to create a new message definition from a COBOL data structure using the New Message Definition File wizard in the workbench.

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Importing file systems into the workbench” on page 124

Before you begin this task, you should be aware of the points listed below:



- To create a new message definition file from a COBOL data structure, the COBOL file must already be present in the workbench, for example in your message set project. This allows you to select the file in the New Message Definition File wizard.
- The wizard allows you to import COBOL files with `.cbl`, `.ccp`, `.cob` and `.cpy` extensions. If your source file has a different extension, you must rename it before attempting to import it.
- If the message set to which you are adding the new message definition file does not have an Custom Wire Format (CWF) layer, only the logical information appears in the model. You can add the physical layer to the message set before or after importing a COBOL data structure but it is recommended that you add the physical layer before you import the data structure.
- It is also possible to import a COBOL data structure from the command line using `mqsicreatemsgdefs`.

If you previously used the WebSphere MQ Integrator Version 2.1 COBOL importer to create a message model, you may find that the WebSphere Message Broker Version 6.0 importer does not produce exactly the same results. In particular, the Version 2.1 COBOL importer did not support as many COBOL constructs as WebSphere Message Broker Version 6.0, and the naming convention for Version 2.1 compound types is not the same as the naming convention for WebSphere Message Broker Version 6.0 complex types and groups. In a very few cases, the naming convention for Version 2.1 elements is not the same as the naming convention for WebSphere Message Broker Version 6.0 elements. If you created message flow ESQL in Version 2.1 to manipulate your COBOL-derived message model, and you have imported that ESQL into WebSphere Message Broker Version 6.0 using the `mqsimigratemsgflows` command, and you are now recreating your message model using the WebSphere Message Broker Version 6.0 COBOL importer, you are strongly recommended to review your ESQL for correctness.

The steps below cover creating a completely new message definition file and overwriting the contents of an existing file.

To create a message definition file from a COBOL data structure:

1. Switch to the Broker Application Development perspective.
2. Open the New Message Definition File wizard by clicking **File > New > Message Definition File** from the workbench menu. Alternatively, you can open the wizard by right-clicking a COBOL copybook previously imported into the workbench and clicking **New > Message Definition File** on the menu.
3. In the displayed list of options, click **COBOL file** then click **Next**.
4. Step through the remainder of the wizard filling in the details as required.

When you have completed importing the COBOL file using the wizard:

- Carefully check for any errors in the report that is created when the file is imported. You can find this report in the log directory within the project containing the message definition that you have attempted to create. The report has a `.cobol.report.txt` file extension, prefixed with the name that you specified for the new message definition file.
- Review the messages shown in the workbench task list to check whether any new warnings or errors have appeared.

## Importing from the command line

This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML schema in order to populate a message set with message definitions.

### Before you start:

Before you attempt the task described in this topic, it is recommended that you read the following reference topic:

- **mqsicreatemsgdefs** command

The command line importer allows you to create a new message set, into which the message definition files will be placed. When you create a new message set from the command line, only the logical information is created by default. However, the command line importer allows you to create a new message set based on an existing message set. The physical format information from the base message set is also created in the new message set. If you want physical format information to be created as well, you must do the following before you invoke the **mqsicreatemsgdefs** command:

1. Using the workbench, create a message set in your workspace that is to be used as a base message set.
2. To this base message set, add the physical formats that you want to be created in your new message set.

To import C, COBOL copybooks, XML DTD or XML schema using the command line:

1. Close the workbench. This must not be running when you use the command line importer.
2. Invoke the **mqsicreatemsgdefs** command from a command prompt specifying the message set project name, path name of the source files folder, and any other optional parameters that you require. If you want to add physical formats to the new message set that the **mqsicreatemsgdefs** command creates, specify the base message set that contains these physical formats as the **-base** parameter on the import command line.
3. When the command has completed, open **mqsicreatemsgdefs.report.txt**. This report is created when you invoke the **mqsicreatemsgdefs** command and by default is written to the directory from which you invoked the command. The report provides you with the following information:
  - Details of the parameters that were used when **mqsicreatemsgdefs** was invoked.
  - The message set level action.
  - The name of the file or files that have been imported.
  - Details of the import process (for example, any warnings that have been generated and message model objects that have been created).
  - The number of files imported.
4. Start the workbench and switch to the Broker Application Development perspective. The message definition file that was created when you invoked **mqsicreatemsgdefs** is visible in the project that you specified.

If an error occurs during the import of a C, COBOL copybook, XML DTD, or XML schema file, carefully check any errors that the importer reports. By default, all errors are written to the screen and to the log file described above. To gather

additional information about the import, specify the -v (Verbose) command line parameter. This parameter displays more detailed information as the import proceeds.

## Importing from IBM supplied messages

This describes how to create a new message definition from IBM supplied messages using the New Message Definition File wizard in the workbench.

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81

The following steps cover creating a completely new message definition file and also overwriting the contents of an existing file.

To create a message definition from an IBM supplied message:

1. Switch to the Broker Application Development perspective.
2. Open the New Message Definition File wizard by clicking **File > New > Message Definition File** on the workbench menu.
3. In the displayed list of options, click **IBM supplied message** to select it then click **Next**.
4. Step through the remainder of the wizard filling in the details as required.

### Note:

- A number of the panels and options could be subject to settings you make within the wizard and it is possible that they will not always appear.
- Some fields in the wizard could be greyed out. This is likely to be due to the field only having one option, a mandatory setting, or is not used due to other settings that have already been made.

When you have completed the IBM supplied message import using the wizard:

- Carefully check for any errors in the report that is created when the file is imported. You can find this report in the log directory within the project containing the message definition that you have attempted to create. The report has a `.xsd.report.txt` file extension, prefixed with the name that you specified for the new message definition file.
- Review the messages shown in the workbench task list to check whether any new warnings or errors have appeared.

## Importing from WSDL

This describes how to create a new message definition from WSDL using the New Message Definition File wizard in the workbench.

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Importing file systems into the workbench” on page 124

Before you begin this task you should note the following points:

- To ensure the highest interoperability of your web services you are recommended to:
  - Use the document style of WSDL whenever possible
  - Where the use of rpc style WSDL is necessary, use literal encoding
  - Where the use of rpc style with SOAP encoding is necessary, ensure that the WSDL operation definitions are qualified by a valid namespace attribute
- WSDL definitions can optionally be split into multiple files. The typical arrangement would be that a top-level service definition file imports a binding file, the binding file imports a message definition file and this finally imports or includes schema definition files.
- When you create a new message definition file from a WSDL file, the WSDL file must already be present in your message set project. If the WSDL is split into multiple files as described above then you can import either the top-level service definition file or the binding file. In this case the file you import and any files it refers to using a relative location must be in the workspace before the WSDL import is run.
- Where you are importing two or more WSDL files that share common schema definitions, you are recommended to import them into separate message set projects to avoid the likelihood of any name clashes.
- You should add an XML wire format layer and enable namespace support for the message set that you are adding the new message definition file to. In order to improve web services interoperability, you are recommended to avoid unnecessary customization of the XML wire format layer for messages participating in web services processes.

**Important:** Note that the extension to the file must be `.wsdl` in lower case.

The following steps cover creating a completely new message definition file and also overwriting the contents of an existing file.

To create a message definition from a WSDL file or files:

1. Switch to the Broker Application Development perspective.
2. Open the New Message Definition File wizard by clicking **File > New > Message Definition File** on the workbench menu. Alternatively, you can open the wizard by right-clicking a `.wsdl` file previously imported into the workbench and clicking **New > Message Definition File** on the menu.
3. In the displayed list of options, click **WSDL file** to select it then click **Next**.
4. Step through the remainder of the wizard filling in the details as required.

**Note:**

- A number of the panels and options could be subject to settings you make within the wizard and it is possible that they will not always appear.
- Some fields in the wizard could be greyed out. This is likely to be due to the field having a mandatory setting, or is not used due to other settings that have already been made.

When you have completed the WSDL import using the wizard:

- Carefully check for any errors in the report that is created when the file is imported. You can find this report in the log directory within the project

containing the message definition that you have attempted to create. The report has a `.wsdl.report.txt` file descriptor, prefixed with the name that you specified for the new message definition file.

- Review the messages shown in the workbench task list to check whether any new warnings or errors have appeared.

**Note:** Any required SOAP Envelope and SOAP encoding message definitions are automatically added to your message set during the import. If required, you can also import these manually via the New Message Definition File wizard by selecting the new option **IBM supplied message**.

## Importing WSDL definitions from the command line

This describes how to use the WSDL command line importer (`mqsicreatemsgdefsfromwsdl`) to populate a message set with message definitions.

### Before you start:

Before you attempt the task described in this topic, it is recommended that you read the following reference topic:

- `mqsicreatemsgdefsfromwsdl` command.

The WSDL command line importer allows you to create a new namespace enabled message set, into which the message definition files will be placed, or to add message definition files to an existing namespace enabled message set. If the message set already exists, it must be namespace enabled.

When you create a new message set from the command line, only the logical information is created by default. If you require physical formats in the message set you have two options:

- The command line importer allows you to create a new message set based on an existing message set. The physical format information from the base message set is also created in the new message set. If you want physical format information to be created as well, you must do the following before you invoke the `mqsicreatemsgdefsfromwsdl` command:
  1. Using the workbench, create a message set in your workspace that is to be used as a base message set.
  2. To this base message set, add the physical formats that you want to be created in your new message set.
- Use the workbench to create or open the message set and directly add the physical formats to the message set prior to importing the WSDL definitions into it.

The `mqsicreatemsgdefsfromwsdl` command copies the WSDL files it needs into the workspace before the import runs. These are the top level WSDL file and any further files that may be linked to from it. The files are copied under the specified message set in a folder called `importFiles` and are not removed after the import, allowing the user to subsequently update or run validation on them in the workbench.

To import WSDL definitions using the command line:

1. Close the workbench. This must not be running when you use the command line importer.
2. Invoke the `mqsicreatemsgdefsfromwsdl` command from a command prompt specifying the message set project name, the path name of the directory where

the top level WSDL file is located, the name of that file, and any other optional parameters that you require. If you want to add physical formats to the new message set that the **mqsicreatemsgdefsfromwsdl** command creates, specify the base message set that contains these physical formats as the **-base** parameter on the import command line.

3. When the command has completed, check log file. It has the file descriptor `*.wsdl.report.txt` and the name will be the one you specified in the command. This report is created when you invoke the **mqsicreatemsgdefsfromwsdl** command and by default is written to the directory from which you invoked the command. The report provides you with the following information:
  - Details of the parameters that were used when **mqsicreatemsgdefsfromwsdl** was invoked.
  - The name of the file that has been imported.
  - Details of the import process (for example, any warnings that have been generated and message model objects that have been created).
4. Start the workbench and switch to the Broker Application Development perspective. The message definition file that was created when you invoked **mqsicreatemsgdefsfromwsdl** is visible in the project that you specified.

If an error occurs during the import of a WSDL definition, carefully check any errors that the importer reports. By default, all errors are written to the screen and to the file described above. To gather additional information about the import, specify the **-v** (Verbose) command line parameter. This parameter displays more detailed information as the import proceeds.

## Importing from XML DTD

This describes how to create a new message definition from an XML DTD using the New Message Definition File wizard in the workbench.

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Importing file systems into the workbench” on page 124

Before you begin this task, you should be aware of the points listed below:

- To create a new message definition file from an XML DTD, the DTD file must already be present in the workbench, for example in your message set project. This allows you to select the DTD file in the New Message Definition File wizard.
- If the message set to which you are adding the new message definition file does not have an XML wire format (XML) layer only the logical information appears in the model. You can add the physical layer to the message set before or after importing from a XML DTD but is recommended that you add the physical layer before importing.
- It is also possible to import an XML DTD from the command line using **mqsicreatemsgdefs**.
- The file extension must be `.dtd` in lower case.

The following steps cover both creating a completely new message definition file and overwriting the contents of an existing file.

To create a message definition from an XML DTD:

1. Switch to the Broker Application Development perspective.
2. Open the New Message Definition File wizard by clicking **File > New > Message Definition File** from the workbench menu.
3. In the displayed list of options, click **XML DTD file** to select it then click **Next**.
4. Step through the remainder of the wizard filling in the details as required.

When you have completed importing the XML DTD using the wizard:

- Carefully check for any errors in the report that is created when the file is imported. You can find this report in the log directory within the project containing the message definition that you have attempted to create. The report has a `.dtd.report.txt` file extension, prefixed with the name that you specified for the new message definition file.
- Review the messages shown in the workbench task list to check whether any new warnings or errors have appeared.

The message definition file is created from the XML DTD and is opened in the Message Definition editor so that you can check the imported information and make any required changes. While you are checking the newly created message definition file, review any messages that appear in the workbench task list to see whether you need to make any corrections to resolve errors or warnings relating to the new file.

## Importing from the command line

This describes how to use the command line importer `mqsicreatemsgdefs` to import C, COBOL copybooks, XML DTD or XML schema in order to populate a message set with message definitions.

### Before you start:

Before you attempt the task described in this topic, it is recommended that you read the following reference topic:

- `mqsicreatemsgdefs` command

The command line importer allows you to create a new message set, into which the message definition files will be placed. When you create a new message set from the command line, only the logical information is created by default. However, the command line importer allows you to create a new message set based on an existing message set. The physical format information from the base message set is also created in the new message set. If you want physical format information to be created as well, you must do the following before you invoke the `mqsicreatemsgdefs` command:

1. Using the workbench, create a message set in your workspace that is to be used as a base message set.
2. To this base message set, add the physical formats that you want to be created in your new message set.

To import C, COBOL copybooks, XML DTD or XML schema using the command line:

1. Close the workbench. This must not be running when you use the command line importer.

2. Invoke the **mqsicreatemsgdefs** command from a command prompt specifying the message set project name, path name of the source files folder, and any other optional parameters that you require. If you want to add physical formats to the new message set that the **mqsicreatemsgdefs** command creates, specify the base message set that contains these physical formats as the **-base** parameter on the import command line.
3. When the command has completed, open `mqsicreatemsgdefs.report.txt`. This report is created when you invoke the **mqsicreatemsgdefs** command and by default is written to the directory from which you invoked the command. The report provides you with the following information:
  - Details of the parameters that were used when **mqsicreatemsgdefs** was invoked.
  - The message set level action.
  - The name of the file or files that have been imported.
  - Details of the import process (for example, any warnings that have been generated and message model objects that have been created).
  - The number of files imported.
4. Start the workbench and switch to the Broker Application Development perspective. The message definition file that was created when you invoked **mqsicreatemsgdefs** is visible in the project that you specified.

If an error occurs during the import of a C, COBOL copybook, XML DTD, or XML schema file, carefully check any errors that the importer reports. By default, all errors are written to the screen and to the log file described above. To gather additional information about the import, specify the **-v** (Verbose) command line parameter. This parameter displays more detailed information as the import proceeds.

## Importing from XML schema

This describes how to create a new message definition from an XML Schema using the New Message Definition File wizard in the workbench.

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Importing file systems into the workbench” on page 124

Before you begin this task, you should be aware of the points listed below:

- To create a new message definition file from an XML schema, the schema file must already be present in the workbench, for example in your message set project. This allows you to select the schema file in the New Message Definition File wizard.
- If the message set to which you are adding the new message definition file does have an XML wire format layer, but no namespace support, the imported schema is modified to remove namespaces. It is therefore recommended that you enable namespace support before importing a schema.
- If the message set to which you are adding the new message definition file does not have an XML wire format layer, but does have namespace support, only the logical information appears in the model. The XML schema is not modified to remove namespaces. You can add the physical layer to the message set before or



after importing the XML schema but it is recommended that you add the physical layer before importing a schema.

- If the message set to which you are adding the new message definition file does not have an XML wire format layer, and does not have namespace support, only the logical information appears in the model and the imported schema is modified to remove namespaces.
- If you are working with a message set that does not have namespace support, you must specify the preferences that apply when you import a schema into the message set. These preferences allow you to specify how the importer treats certain individual schema constructs. You can either reject the schema if any occurrences of the construct are encountered or modify occurrences of the construct. If you choose modify, the importer modifies all occurrences of the construct.
- The extension to the XML Schema file must be `.xsd` in lower case.

The following steps cover creating a completely new message definition file and also overwriting the contents of an existing file.

To create a message definition from an XML schema file:

1. Switch to the Broker Application Development perspective.
2. Open the New Message Definition File wizard by clicking **File > New > Message Definition File** on the workbench menu. Alternatively, you can open the wizard by right-clicking an `*.xsd` file previously imported into the workbench and clicking **New > Message Definition File** on the menu.
3. In the displayed list of options, click **XML schema file** to select it then click **Next**.
4. Step through the remainder of the wizard filling in the details as required.

When you have completed importing the XML Schema using the wizard:

- Carefully check for any errors in the report that is created when the file is imported. You can find this report in the log directory structure within the project containing the message definition that you have attempted to create. The report has a `.xsd.report.txt` file extension, prefixed with the name that you specified for the new message definition file.
- Review the messages shown in the workbench task list to check whether any new warnings or errors have appeared. Some examples of situations where messages appear are given below:
  - If the XML schema that you are importing contains `xsd:key`, `xsd:keyref` and `xsd:unique` constructs, warning messages appear in the task list to tell you that these constructs are unsupported and will be ignored by the broker. If you prefer to delete these constructs, open the message definition file in the Message Definition editor, and delete the constructs as described in “Deleting objects” on page 117. Deleting the constructs also removes the warning messages from the task list. If you decide not to delete the constructs, they will remain in the message model but will not be deployed to the broker, or be used for any other purpose. The warning messages in the task list will remain, but you will be able to use the message model normally.
  - If the XML schema that you are importing contains `xsd:redefine` constructs, error messages appear in task list to tell you that this construct is unsupported. If you right-click on the error messages and select **Quick Fix** you can choose to convert the `xsd:redefine` constructs into `xsd:include` constructs. This also removes the error messages.

- If you are importing a collection of related XML schema files and the Message Definition Editor is unable to resolve the links between two of the imported files, messages appear in the task list to say that referenced types or other objects cannot be found. If this occurs, refer to Resolving problems when developing message models for further information.

## Importing from the command line

This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML schema in order to populate a message set with message definitions.

### Before you start:

Before you attempt the task described in this topic, it is recommended that you read the following reference topic:

- **mqsicreatemsgdefs** command

The command line importer allows you to create a new message set, into which the message definition files will be placed. When you create a new message set from the command line, only the logical information is created by default. However, the command line importer allows you to create a new message set based on an existing message set. The physical format information from the base message set is also created in the new message set. If you want physical format information to be created as well, you must do the following before you invoke the **mqsicreatemsgdefs** command:

1. Using the workbench, create a message set in your workspace that is to be used as a base message set.
2. To this base message set, add the physical formats that you want to be created in your new message set.

To import C, COBOL copybooks, XML DTD or XML schema using the command line:

1. Close the workbench. This must not be running when you use the command line importer.
2. Invoke the **mqsicreatemsgdefs** command from a command prompt specifying the message set project name, path name of the source files folder, and any other optional parameters that you require. If you want to add physical formats to the new message set that the **mqsicreatemsgdefs** command creates, specify the base message set that contains these physical formats as the **-base** parameter on the import command line.
3. When the command has completed, open **mqsicreatemsgdefs.report.txt**. This report is created when you invoke the **mqsicreatemsgdefs** command and by default is written to the directory from which you invoked the command. The report provides you with the following information:
  - Details of the parameters that were used when **mqsicreatemsgdefs** was invoked.
  - The message set level action.
  - The name of the file or files that have been imported.
  - Details of the import process (for example, any warnings that have been generated and message model objects that have been created).
  - The number of files imported.

4. Start the workbench and switch to the Broker Application Development perspective. The message definition file that was created when you invoked `mqsicreatemsgdefs` is visible in the project that you specified.

If an error occurs during the import of a C, COBOL copybook, XML DTD, or XML schema file, carefully check any errors that the importer reports. By default, all errors are written to the screen and to the log file described above. To gather additional information about the import, specify the `-v` (Verbose) command line parameter. This parameter displays more detailed information as the import proceeds.

---

## Generating information from message sets

This topic area describes how to generate information from a message set.

### Before you start:

The tasks in this topic area assume that you have already designed and constructed your message model.

When you have designed and constructed your message model you can generate information about a message set. The following tasks relate to generating information from message sets:

- “Generating message set documentation as HTML”
- “Generating XML schema” on page 140
- “Generating a Web Service Definition from a message set” on page 141

## Generating message set documentation as HTML

This task topic describes how to generate message set documentation as HTML.

To generate message set information as HTML:

1. Switch to the Broker Application Development perspective.
2. Select the message set from which you want to generate documentation as HTML.
3. From the workbench menu, click **File > New > Message Set Documentation** to open the New Message Set Documentation wizard. Alternatively, right-click anywhere in the Resource Navigator then click **New > Message Set Documentation** on the pop-up menu.
4. The message set that you selected is highlighted in the wizard. If this is not the message set from which you want to generate HTML, select the correct message set.
5. Click **Next** to move to the next page of the wizard.
6. Click the destination folder for the HTML documentation.

**Tip:** The folder that you select must be within the project containing the message set from which you are generating the documentation. The documentation that you generate is automatically placed in a folder with the same name as the message set. If the destination folder that you choose already exists, the contents are overwritten.

7. Create a new folder for the documentation by carrying out the following steps:
  - a. Click the location for the new folder in the project hierarchy.
  - b. Click **Create New Folder** and type a name for the new folder.

- c. Click **OK**.
8. Click **Finish**. Your HTML documentation is generated.
9. View the generated documentation by carrying out the following steps:
  - a. In the Resource Navigator, expand the package that has exactly the same name as the message set from which you generated the documentation. This package is created as part of the documentation generation process.
  - b. In the package, right-click the file `index.html`, then click **Open With > System Editor** on the pop-up menu. The generated HTML documentation opens in your system's default web browser.

## Generating XML schema

### Before you start:

To complete this task, you must have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Working with a message definition file” on page 91
- “Working with message model objects” on page 92

**Tip:** You are recommended to replace any deprecated constructs before generating XML schema representations of your models.

This task topic describes how to generate XML schema from a message definition file:

1. Switch to the Broker Application Development perspective.
2. Select the message definition file (\*.msxd file extension) from which you want to generate XML schema.
3. From the workbench menu, click **File > New > XML Schema** to open the New XML Schema wizard. Alternatively, right-click the message definition file from which you want to generate XML schema then click **New > XML Schema** on the pop-up menu.
4. The message definition file that you selected is highlighted. If this is not the message definition file from which you want to generate XML schema, select the correct message definition file.
5. Optional: From the drop down list, at the bottom of the New XML Schema page, click the XML Wire Format that you want to use to generate XML schema.

**Tip:** You must have previously added one or more XML Wire Format layers to a message set if you want to use an XML physical format when you generate XML schema. For further information see “Adding an XML wire format” on page 85.

6. If you do not want strict generation of XML schema, clear the **Strict generation** check box at the bottom of the New XML Schema page. By default, this check box is selected.

**Tip:** For further information on strict and lax generation of XML schema, see “Generate XML Schema” on page 76.

7. Click **Next** to move to the next page of the wizard.
8. Select a destination folder for the XML schema.

**Tip:** The folder that you select must be within the project containing the message definition file from which you are generating the documentation. If a folder is not selected, the generated XML schema is placed in the project folder by default. Note that the contents of the folder that you select are overwritten.

9. Create a new folder for the documentation by carrying out the following steps:
  - a. Click the location for the new folder in the project hierarchy.
  - b. Click **Create New Folder** and type a name for the new folder.
  - c. Click **OK**.
10. Click **Finish**. Your XML schema is generated.
11. Use the Resource Navigator to locate the destination folder that you specified for the generated XML schema. This folder contains a file with exactly the same name as your message definition file with the file extension \*.xsd. This is the generated XML schema. To view this file, right-click it then click **Open** on the pop-up menu. This launches the schema editor.

**Tip:** The **Design**, **Source** or **Graph** tabs located in bottom left corner of the schema editor provide you with different views of generated XML schema.

## Generating a Web Service Definition from a message set

This describes how to generate a Web Services Description Language (WSDL) document from a message set, that specifies the interface for a Web Service.

### Before you start:

To complete this task you must already have completed the following tasks:

- “Creating a message set project” on page 79
- “Creating a message set” on page 81
- “Creating a message definition file” on page 91
- “Creating a message category file” on page 120

Before you begin this task you should note the following points:

- To ensure the highest interoperability of your web services you are recommended to:
  - Use the document style of WSDL whenever possible
  - Where the rpc style WSDL is necessary, use literal encoding
- You are recommended to replace any deprecated constructs before generating WSDL representations of your message models.

To generate a Web Service Definition:

1. Switch to the Broker Application Development perspective.
2. In the Resource Navigator, right-click the message definition file from which you want to generate a web service definition and select **New** → **WSDL Definition**. Alternatively, open the wizard by right-clicking anywhere in the Resource Navigator then clicking **New** → **WSDL Definition** on the pop-up menu, or by clicking **File** → **New** → **WSDL Definition** on the workbench menu. This will start the New WSDL Definition wizard.
3. Step through the wizard filling in the details as required.

**Note:**

- A number of the panels and options are subject to settings you make within the wizard and it is possible that they will not always appear.
- Some fields in the wizard could be greyed out. This is likely to be due to the field having a mandatory setting, or is not used due to other settings that have already been made.

On completion of the New WSDL Definition wizard, you will have generated a WSDL definition. The file extension for WSDL files is \*.wsdl, and for any imported schema files in multi-file mode (where the WSDL definition is split over a number of files) is \*.xsd.

This following is an example of the expected WSDL generated for a JMS binding:

```
<wsdl:service name='HTTP'>
  <wsdl:port binding='tns:HTTPSoapBinding' name='HTTP'>
    <wsdl:soap:address
      location='jms:/queue?destination=jms/MyQueue&
        connectionFactory=jms/MyCF&
        priority=5&
        targetService=GetQuote' />
    </wsdl:port>
  </wsdl:service>
```

**Note:** The various parts of the location string are broken over separate lines for clarity, but are actually generated as a continuous string without additional white space.

---

## Part 2. Reference

<b>Message model reference information</b> . . . . .	145
Message set preferences . . . . .	145
Editors . . . . .	145
Validation . . . . .	146
XML Schema Importer . . . . .	147
Message set properties . . . . .	147
Custom Wire Format message set properties . . . . .	150
XML Wire Format message set properties . . . . .	155
TDS Format message set properties . . . . .	162
Documentation properties for a message set . . . . .	172
Message definition file properties. . . . .	174
Message definition file includes properties . . . . .	175
Message definition file imports properties . . . . .	175
Message definition file redefines properties . . . . .	175
Documentation properties for all message set objects. . . . .	175
Message category properties . . . . .	175
Message category member properties . . . . .	176
Message model object properties . . . . .	177
Logical properties for message model objects . . . . .	177
Physical properties for message model objects . . . . .	203
Documentation properties for all message set objects. . . . .	233
Message model object properties by object. . . . .	233
Deprecated message model object properties . . . . .	549
Logical properties for deprecated message model objects . . . . .	549
Physical properties for deprecated message model objects . . . . .	552
Documentation properties for all message set objects. . . . .	556
Deprecated message model object properties by object . . . . .	556
Additional MRM domain information . . . . .	673
Additional logical information. . . . .	673
Additional CWF information . . . . .	675
Additional XML information . . . . .	676
Additional TDS information . . . . .	679
DateTime formats . . . . .	701
Message model task list errors that have a quick fix . . . . .	710
Additional MIME domain information . . . . .	712
MIME standard header fields . . . . .	712
MIME Parser Use and Restrictions . . . . .	715
Generated model representations . . . . .	716
Document generation. . . . .	716
WSDL generation . . . . .	716
XML Schema generation. . . . .	719
Import formats . . . . .	722
Importing from C: supported features . . . . .	722
Importing from COBOL: supported features . . . . .	724
Importing from WSDL: generated objects and restrictions . . . . .	729
Importing from XML Schema: unsupported features . . . . .	732
Importing from XML DTD: unsupported features . . . . .	734
Message model wizards . . . . .	734
New message definition file wizard . . . . .	735
New WSDL definition wizard . . . . .	764





---

## Message model reference information

Message model reference information is available for:

- “Message set preferences”
- “Message set properties” on page 147
- “Message definition file properties” on page 174
- “Message category properties” on page 175
- “Message model object properties” on page 177
- “Deprecated message model object properties” on page 549
- “Additional MRM domain information” on page 673
- “Additional MIME domain information” on page 712
- “Generated model representations” on page 716
- “Import formats” on page 722
- “Message model wizards” on page 734

---

## Message set preferences

Property	Type	Meaning
Default version tag	String	Provide the default version information you would like to be set in the message set Version property when you create a new message set.

You can alter a number of the preferences that affect the way certain areas of message set processing are handled. The areas are:

- “Editors”
- “Validation” on page 146
- “XML Schema Importer” on page 147

## Editors

### Message set editor settings

Outline view settings.

While looking at a large message set containing a number of message definition files that have different namespaces or multiple message definition files that have same namespace, you might want to view the information in alternate ways to make it easier for you to visualize the structure of the message set. If you double click on the global construct, it will open message definition file in which the global construct is defined.

Property	Type	Meaning
Group by namespace and then by collections	Button	Selecting this view groups the global constructs by namespace and then by collection (for example, Messages, Types, Groups, or Elements and Attributes). Using this view you can visualize all of the constructs that belong to each of the defined namespaces.

Property	Type	Meaning
Group by collections and then by namespace	Button	Selecting this view groups the global constructs by collection (for example, Messages, Types, Groups, or Elements and Attributes) and then by namespace. Using this view you can visualize which global construct in the message set is defined in which namespace.

### Message definition editor settings

Property	Type	Meaning
Show base complex types	Check box	Where your complex type is based on another complex type that is derived by an extension, selecting this will display the base complex type in the outline view.

## Validation

You can customize a number of the warning messages that the message set validation will generate. This is done through the Message Set Validation Preference page.

Any warning or error that falls into any of the following categories below can be customized on a category basis. The customization consists of both severity and priority. The severity can be any of the following:

- Error
- Warning
- Info
- Ignore

The priority can be any of the following as long as severity is not Ignore:

- High
- Normal
- Low

If severity is Ignore the user will not be able to change the priority.

### Message set validation settings

The following is a list of the categories that can be customized;

- Use of deprecated constructs
- Messages with abstract global elements
- Facet runtime validation differences
- Type/Element substitution runtime validation differences
- Mixed content runtime validation differences
- Wildcard runtime validation differences
- Tagged/Delimited String group content
- Zero Custom Wire Format length count
- Zero Tagged/Delimited String Format length count
- Empty Tagged/Delimited String Format tag
- List or Union with Custom Wire Format
- List or Union with Tagged/Delimited String Format

- Unbounded max occurs with Custom Wire Format
- Unbounded max occurs with Tagged/Delimited String Format

## XML Schema Importer

You can customize the following categories that affect the way in which an XML schema is imported into a message set that does not support namespaces.

Category	Modify	Reject	Accept
Import	Converts Import to Include	Import fails if it sees an Import	Not applicable
Redefine	Removes the Redefine statements	Import fails if it sees a Redefine	Redefine imported (gives task list error)
List	Changes type base to xsd:string	Import fails if it sees a List	List imported
Union	Changes type base to xsd:string	Import fails if it sees a Union	Union imported
Abstract Complex Type	Sets abstract to false	Import fails if it sees an Abstract Complex Type	Abstract Complex Type imported
Abstract Element	Sets abstract to false	Import fails if it sees an Abstract Element	Abstract Element imported

---

## Message set properties

### General message set properties

The table below defines the properties that you can set to customize the message set.

Property	Type	Meaning
Message set ID	String	This is a unique identifier that is automatically generated for you when you create the message set. You cannot change this property.
Message Set Alias	String	Specify an alternative unique value that identifies the message set. This property is only required if you are using the Message Identity technique to identify embedded messages. This is where the embedded messages are defined in this message set but the parent message is defined in a different message set, and the bit stream does not contain the actual message set name or identifier
Default Wire Format	String	Specify the default wire format that is used if a format cannot be deduced from the message's MQRFH2 header, or was not specified as a property of the input node at which the message is received by a message flow. The default is empty (not set).

Property	Type	Meaning
Message Type Prefix	String	<p>This property is used when you define multipart messages, specifically when using the Message Path technique to identify embedded messages.</p> <p>The value that you specify is used as an absolute or relative path to the innermost message from the outermost, and is used as a prefix to the value of the <b>Message Type</b> property specified for the outermost message (specified either in the MQRFH2 header of the message, or in the input node of the message flow).</p> <p>If you set a value, it must be in the form id1/id2/.../idn where id1 is the identifier of the outermost message, id2 is the identifier of the next element or message, and idn is the identifier of the innermost message. The default value is blank (not set).</p> <p>The table below, describing the use of the message set property <i>Message Type Prefix</i>, shows how this value is combined with the <i>Message Type</i> property of an input message.</p>
Runtime Parser	String	<p>Select the message parser for messages belonging to this set from the drop-down list. The message parser name should be used when writing ESQL field references for messages in the message set, for example, InputRoot.MRM.Document. The Mapping editor and the content assist feature of the ESQL editor use the message parser name when generating ESQL field references. The message parser name must match the Message Domain property of any input node that processes messages from the message set, or the &lt;Msd&gt; element value of any MQRFH2 header preceding a message from the message set. You can choose from:</p> <ul style="list-style-type: none"> <li>• MRM (the default). If you specify MRM, you must deploy the message set to the brokers that receive these messages. The deploy action creates a runtime dictionary (RTD) against which the MRM parser invoked by the broker checks the received message.</li> <li>• XML, XMLNS, XMLNSC, JMSMap, or JMSStream. Choose one of these parsers if you want to model a generic self-defining XML, XMLNS, XMLNSC, JMSMap, or JMSStream message. Messages defined in this way are interpreted by the generic XML parser, not the MRM parser. (You do not have to deploy these message sets to brokers, because these parsers do not use a runtime dictionary.)</li> <li>• IDOC. Choose this parser if you want to model SAP IDoc messages. If your message set contains user-defined IDoc segments then you must deploy the message set to the brokers that receive these messages. The deploy action creates a runtime dictionary (RTD) against which the MRM parser that the broker invokes checks the received message.</li> <li>• MIME. Choose this parser if you want to model a MIME message. Messages defined in this way are interpreted by the MIME parser, not the MRM parser. (You do not have to deploy these message sets to brokers, because this parser does not use a runtime dictionary.)</li> </ul>

Property	Type	Meaning
Use namespaces	Check box	<p>Select this property if you want to use namespaces within the message set. Namespaces provide a method of avoiding naming conflicts where different document definitions have elements of the same name. For further information see Namespaces.</p> <p>Once this property has been enabled, it cannot be disabled.</p> <p>Note that using namespaces also affects how elements are created in the WebSphere Message Broker logical message tree. Each element in the message tree has both a name and a namespace, so an ESQL or Java reference to one of these elements has to specify both name and namespace. Therefore using namespaces has an effect on the ESQL or Java you write.</p> <p>Select this property if the message set will be used now or in the future to model XML messages. However, if the message set is to be used with message flows running on WebSphere Message Broker Version 2.1, or that originated on WebSphere Message Broker Version 2.1, you should <i>not</i> select this property, because such message flows are not namespace aware.</p>
Treat Length facet as MaxLength	Check box	<p>Select this property if you want the COBOL importer to create a maxLength facet for a fixed length string element rather than a length facet.</p> <p>The default is for this property to be set.</p>

### Use of the message set property Message Type Prefix

The table below shows the implications of using the property *Message Type Prefix*. Note that what the message type or message prefix may describe could be elements or messages.

Message Type property example	Message Type Prefix not set	Message Type Prefix set
Simple Message Type:msg_type	Results in the simple Message Type:msg_type	Results in the path Message Type: /msg_prefix_1/.../msg_prefix_n/ msg_type
Path Message Type:msg_type_1/.../msg_type_m	Results in the path Message Type:/msg_type_1/.../msg_type_m	Results in the combined path Message Type: /msg_prefix_1.../msg_prefix_n /msg_type_1/.../msg_type_m
Simple absolute Message Type:/msg_type	Results in the simple Message Type:msg_type	Results in the simple Message Type:msg_type  An error is raised if Message Type Prefix is set to any value other than msg_type.
Path absolute Message Type:/msg_type_1/.../msg_type_m	Results in the path Message Type:/msg_type_1/.../msg_type_m	Results in the path Message Type:/msg_type_1/.../msg_type_m  An error is raised if all identifiers in Message Type Prefix do not match the corresponding identifiers in the resulting path.

The message set does not have any properties that are dictated by membership of a larger object, because this is the largest message object as defined by the MRM.

In addition to message set properties, each of the physical formats have properties that can be updated. There is also a documentation property for a message set. Details of these can be found at;

- “Custom Wire Format message set properties”
- “XML Wire Format message set properties” on page 155
- “TDS Format message set properties” on page 162
- “Documentation properties for all message set objects” on page 175

## Custom Wire Format message set properties

The table below defines the properties that you can set for the message set. Some of the message set properties (marked with an asterisk (\*)) are relevant only if the message being processed is *not* using WebSphere MQ as the transport protocol. If the transport protocol is WebSphere MQ, values are derived from the message headers (for example, MQMD), and the message set properties, if set, are ignored.

### Binary representation of boolean values

Property	Type	Meaning
Boolean True Value	STRING	Enter up to eight hexadecimal digits. Do not include the hexadecimal indicator (0x) preceding this number. Each digit is a half byte. The maximum length is 4 bytes. You must enter an even number of digits (a whole number of bytes). This value must be different from, but the same length as, the <i>Boolean False Value</i> . The default value is 00000001.
Boolean False Value	STRING	Enter up to eight hexadecimal digits. Do not include the hexadecimal indicator (0x) preceding this number. Each digit is a half byte. The maximum length is 4 bytes. You must enter an even number of digits (a whole number of bytes). This value must be different from, but the same length as, the <i>Boolean True Value</i> . The default value is 00000000.
Boolean Null Value	STRING	Enter up to eight hexadecimal digits. Do not include the hexadecimal indicator (0x) preceding this number. Each digit is a half byte. The maximum length is 4 bytes. You must enter an even number of digits (a whole number of bytes). This value can be the same as either <i>Boolean True Value</i> or <i>Boolean False Value</i> , or different. The default value is 00000000.

### Output settings

These settings are used when messages are being serialized.

Property	Type	Meaning
Byte Alignment Pad	String	<p>If the xsd:element Custom Wire Format properties Byte Alignment, Leading Skip Count, and Trailing Skip Count cause bytes to be skipped in the bit stream when the message is serialized, this property supplies the character to be used in the skipped positions. Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select SPACE (the default), NUL, or 0 from the drop-down list.</li> <li>• Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a decimal character code in the form YY where YY is a decimal value.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.</li> </ul>

Property	Type	Meaning
Policy for Missing Elements	Enumerated Type	<p>The setting of this property governs the action taken by the broker when fields are missing from the message tree when the message is serialized:</p> <ul style="list-style-type: none"> <li>• Use <i>Default Value</i> (the default). If a <i>Default Value</i> exists for the element, output it; otherwise throw an exception.</li> <li>• Use <i>Null Value</i>. If the <i>Nillable</i> property of the element is checked, and an <i>Encoding Null Value</i> is specified for the element, output the <i>Encoding Null Value</i> according to the rules defined by the <i>Encoding Null</i> property. Otherwise an exception is thrown.</li> </ul>

## Binary representation of decimal values

Property	Type	Meaning
Packed Decimal Positive Code	Enumerated Type	Select the positive sign used for packed decimal numbers from the drop-down list. The default is C, which indicates that 0x0C is used as the positive sign, which is the usual value. You can also select F, which indicates that 0x0F is used as the positive sign: some systems require this setting.

## DateTime settings

Property	Type	Meaning
Derive default dateTime format from logical type	Button	<p>Select this option if you want the default dateTime format to be determined by the logical type of the element or attribute. The defaults are:</p> <p><b>dateTime</b>      yyyy-MM-dd'T'HH:mm:ssZZZ</p> <p><b>date</b>          yyyy-MM-dd</p> <p><b>time</b>          HH:mm:ssZZZ</p> <p><b>gYear</b>        yyyy</p> <p><b>gYearMonth</b>    yyyy-MM</p> <p><b>gMonth</b>        --MM--</p> <p><b>gMonthDay</b>    --MM-dd</p> <p><b>gDay</b>         ---dd</p> <p>You can override this property for an element or attribute within a complex type.</p>
Use default dateTime Format	Button and dateTime	<p>Select this option if you want to specify a default dateTime format that is fixed for all elements or attributes of logical type dateTime, date, time, gYear, gYearMonth, gMonth, gMonthDay and gDay.</p> <p>The initial value for this property is yyyy-MM-dd'T'HH:mm:ssZZZ, which you can change by over-typing.</p> <p>You can override this property for an element or attribute within a complex type.</p> <p>For more information about dateTime formats, see "DateTime formats" on page 701.</p>

Property	Type	Meaning
Start of century for 2 digit years	Integer	<p>This property determines how two-digit years are interpreted. Specify the two digits that start a 100-year window that contains the current year. For example, if you specify 89, and the current year is 2002, all two-digit dates are interpreted as being in the range 1989 to 2088.</p> <p>The initial value is 53, which you can change by over-typing.</p>
Days in First Week of Year	Enumerated Type	<p>Specify the number of days of the new year that must fall within the first week.</p> <p>The start of a year usually falls in the middle of a week. If the number of days in that week is less than the value specified here, the week is considered to be the last week of the previous year; hence week 1 starts some days into the new year. Otherwise it is considered to be the first week of the new year; hence week 1 starts some days before the new year.</p> <p>Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a number from the drop-down list. The initial value is 4.</p>
First Day of Week	Enumerated Type	<p>Specify the day on which each new week starts.</p> <p>Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a value from the drop-down list. The initial value is Monday.</p>



Property	Type	Meaning
Strict DateTime Checking	Check box	<p>Select this option if you want to restrict dateTimes to a valid dateTime format. This will not allow 35th March to be processed as 4th April, and 10:79 to be processed as 11:19. If <i>Strict DateTime Checking</i> is set, receiving an incorrect dateTime will cause an error. The default is to restrict dateTimes.</p> <p><b>Strict dateTime checking</b> Examples of strict dateTime checking are:</p> <ul style="list-style-type: none"> <li>• DateTimes will be restricted to valid dateTimes only. This will not allow dates such as the 35th March to be processed as 4th April, and 10:79 to be processed as 11:19. Receiving an out-of-band dateTime like these examples, will cause an error to occur.</li> <li>• The number of characters for a numeric dateTime component must be within the bounds of the corresponding formatting symbols. Repeat the symbol to specify the minimum number of digits required. The maximum number of digits permitted will be the upper bound for a particular symbol. For example, day in month has an upper bound of 31 therefore a format string of d will allow the values of 2 or 21 to be parsed but will disallow the value of 32 or 210. On output, numbers are padded with zeros to the specified length. A year is a special case, see message set property of Start of century for 2 digit years. For fractional seconds, the length must implicitly match the number of format symbols on input. The output is rounded to the specified length.</li> <li>• Whitespace will not be skipped over. The whitespace in the input string must correspond with the same number and the position of whitespaces in the formatting string.</li> <li>• If there is still data to be parsed in the input string once all the symbols in the formatting string have been matched, an error will occur.</li> </ul> <p><b>Lenient dateTime checking</b> Examples of lenient dateTime checking are:</p> <ul style="list-style-type: none"> <li>• The parser will convert out-of-band dateTime values to the appropriate in-band value. For example, a date of 2005-05-32 will be converted to 2005-06-01.</li> <li>• Output of dateTimes will always adhere to the symbol count. For example, a formatting string of yyyy-MM-dd (where '-' is the field separator) allows one or more characters to be parsed against MM and dd. This would allow invalid dates such as 2005-1-123 and 2005-011-12 to be input. The first value of 2005-1-123 would be output as the date 2005-05-03, and the second value of 2005-011-12 would be output as the date 2005-11-12.</li> <li>• The number of the timezone formatting symbol Z will only be applicable to the output dateTime format.</li> <li>• Whitespace will be skipped over.</li> </ul>
Time Zone	Enumerated Type	<p>The value that you set for this property is used if the value that you specified for the <i>Default DateTime Format</i> property does not include Time Zone information.</p> <p>The initial value is Use Broker Locale which causes the broker to get the information from the underlying platform.</p> <p>You can change this using the drop down box.</p>

Property	Type	Meaning
Daylight Savings Time	Check box	<p>Select this option if the area in the <i>Time Zone</i> property observes daylight savings time. If it does not observe daylight savings time, this option should not be selected.</p> <p>For example, if an area is selected in <i>Time Zone</i> and this option is not selected, the value passed will represent the time zone without the daylight savings time.</p> <p>Default is not to observe daylight savings time.</p>
Use input UTC format on output	Check box	<p>This property applies to elements and attributes of logical type <code>xsd:dateTime</code> or <code>xsd:time</code> that contain a datetime as a string and that have a dateTime format of I, IU, T or TU or that includes <i>ZZZ</i> or <i>ZZZU</i>.</p> <p>Such elements and attributes can specify Consolidated Universal Time (UTC) by using either the Z character or timezone +00:00 in the value. On input the MRM parser remembers the way that UTC was specified.</p> <p>If this property is checked, and the element or attribute is copied to an output message, then the UTC format will be preserved into the output message and will override that implied by the dateTime format property.</p> <p>If this property is not checked, or the element or attribute was not copied from an input message, then the UTC format in the output message is controlled solely by the dateTime format property.</p> <p>The default for this property is not checked.</p>

## Character and numeric encoding for non-WebSphere MQ messages

These settings are used only for messages with no MQMD.

Property	Type	Meaning
Default CCSID*	INTEGER	<p>Enter a numerical value for the default Coded Character Set Identifier. The default is 500.</p> <p>If the input message is a WebSphere MQ message, the equivalent attribute set for the queue manager is used and this property is ignored.</p>
Byte Order*	Enumerated Type	<p>Select either Big Endian (the default) or Little Endian from the drop-down list to specify the byte order of numbers that are represented as binary integers.</p> <p>In C, this is equivalent to data type short or long. In COBOL, this is equivalent to a PIC 9 COMP, COMP-4, COMP-5 or BINARY data type.</p> <p>Your choice must match the encoding with which messages are created. Big Endian is normally the correct option for messages created on UNIX or z/OS, Little Endian for Windows.</p> <p>This property is not used if the message is received across the WebSphere MQ transport protocol: in this case, the property is deduced from the MQMD of the message, or from the broker queue manager's encoding.</p>
Packed Decimal Byte Order*	Enumerated Type	<p>Select Big Endian (the default) or Little Endian from the drop-down list to specify the byte order of numbers that are represented as packed decimal. In COBOL, this is equivalent to PIC 9 COMP-3 data type. (There is no equivalent data type in C.)</p> <p>Your choice must match the encoding with which messages are created. Big Endian is normally the correct option for messages created on UNIX or z/OS, Little Endian for Windows NT.</p>

Property	Type	Meaning
Float Format*	Enumerated Type	Select one of S390 (the default), IEEE, or Reverse IEEE from the drop-down list to specify the byte order of numbers in the message that are represented as floating point.

## XML Wire Format message set properties

The tables below define the properties for the XML Wire Format for the message set.

### Namespace settings

Property	Type	Meaning
Namespace URI	String	Enter the namespace name that will identify which namespace you are using for the associated prefix.
Prefix	String	Enter the prefix to associate the element and attribute names you use it with to the namespace name.

### Namespace schema locations

Property	Type	Meaning
Namespace URI	String	Enter the namespace name that will identify which namespace you are using.
Schema location	String	Enter the location of the schema for the associated namespace name that will be used to validate objects within the namespace.

### XML declarations

Property	Type	Meaning
Suppress XML Declaration	Check box	Select the check box to suppress the XML declaration. If selected, the declaration (for example, <code>&lt;?xml version='1.0'&gt;</code> ) is suppressed.
XML Version	Enumerated type	This controls the value of the version in the generated XML declaration.  The default is 1.0.  If you set Suppress XML Declaration to Yes, this property is ignored.
XML Encoding	Enumerated type	This controls whether an encoding attribute is written in the generated XML declaration.  If Null is selected, no encoding attribute is written in the XML declaration of the output XML document.  If As document text is selected, an encoding attribute will be generated which is consistent with the text in the XML document.  The default is Null.  If you set Suppress XML Declaration to Yes, this property is ignored.

Property	Type	Meaning
Standalone Document	Enumerated type	<p>Select Yes, No, or Null from the drop-down list. If Null is selected, no standalone declaration is present in the XML declaration. If you select Yes or No, the declaration standalone = "yes" or standalone = "no" is added to the XML declaration when the output message is written.</p> <p>The setting of this property does not determine if an external DTD subset is loaded: external DTD subsets are never loaded in this release.</p> <p>If you set <i>Suppress XML Declaration</i> to Yes, this property is ignored.</p>
Output Namespace Declaration	Enumerated Type	<p>The <i>Output Namespace Declaration</i> property controls where the namespace declarations will be placed in the output XML document. Select from:</p> <ul style="list-style-type: none"> <li>• At start of document. Declarations for all of the entries in the <i>Namespace schema locations</i> table above will be output as attributes of the message in the output XML document. The disadvantage of this option is that in some cases unnecessary declarations may be output.</li> <li>• As required. Declarations will only be output when required by an element or attribute that is in that namespace. The disadvantage of this option is that the same namespace declaration may need to be output more than once in the output XML document.</li> </ul> <p>The default option is At start of document.</p> <p>This property is only active if namespaces are enabled for this message set.</p>

### XML document type settings

Property	Type	Meaning
Suppress DOCTYPE	Check box	<p>If you select the check box, the DOCTYPE (DTD) declaration is suppressed.</p> <p>The default value is checked.</p>
DOCTYPE System ID	String	<p>Specify the System ID for DOCTYPE external DTD subset (if DOCTYPE is present). This is normally set to the name of the generated (or imported) DTD for a message set.</p> <p>If <i>Suppress DOCTYPE</i> is set, this property is ignored and cannot be changed (the field is disabled). The default value is <code>www.mrmnames.net/</code> followed by the message set identifier.</p>
DOCTYPE Public ID	String	<p>Specify the Public ID for DOCTYPE external DTD subset (if DOCTYPE is present, and System ID is specified).</p> <p>If <i>Suppress DOCTYPE</i> is set, this property is ignored and cannot be changed (the field is disabled). The default value is the message set identifier.</p>
DOCTYPE Text	String	<p>Use this property to add additional DTD declarations. It is not parsed by the XML parser and thus it might not be valid XML. You can include ENTITY definitions or internal DTD declarations. It is a string (up to 32KB) in which new line and tab characters are replaced by \n and \t respectively.</p> <p>The content is not parsed, and appears in the output message. If there is an in-line DTD, the content of this property takes precedence.</p> <p>If you have set <i>Suppress DOCTYPE</i>, this property is ignored and cannot be changed (the field is disabled).</p> <p>For more information, see "In-line DTDs and the DOCTYPE text property" on page 162.</p> <p>The default value is empty (not set).</p>

## Root Tag Name

Property	Type	Meaning
Root Tag Name	String	<p>Specify the name of the message set root tag. You can leave this property blank, in which case no wrapper tags are used for messages (that is, the message tag is the root of the document). The name can be followed by a space and additional text for attribute/value pairs to appear with the root tag.</p> <p>The default value is blank.</p>

## Suppress Timestamp Comment

Property	Type	Meaning
Suppress Timestamp Comment	Check box	<p>If selected, the timestamp comment string in the XML output is suppressed.</p> <p>If not selected, the comment is not suppressed, and a comment of the form <code>&lt;!--MRM Generated XML Output on: Tue Apr 23 09:34:42 2002--&gt;</code> is included in the output message.</p> <p>The default is for the check box to be selected.</p>

## Enable Versioning Support

Property	Type	Meaning
Enable Versioning Support	Check box	<p>If this is selected, versioning support is enabled. This property specifies whether XML namespace definitions are coded for the root tag in the message, together with namespace qualifiers for any elements that do not belong to the default namespace. These namespace definitions are used to represent the message set dependency information, which is used to support the exchange of messages between applications that are based on different customizations of the same message set.</p> <p>The default is for the check box to be selected, for compatibility with MRM XML messages in earlier releases. If you did not use MRM XML messages in earlier releases, you are strongly recommended to ensure this check box is not selected.</p>

## XML representation of boolean values

Property	Type	Meaning
Boolean True Value	String	<p>Specify the string that is used to encode and recognize BOOLEAN true values. When an XML document is parsed, the string 1 is always accepted as true for a BOOLEAN element. Enter a string of up to 254 characters.</p> <p>The default is true. 1 is also valid.</p>
Boolean False Value	String	<p>Specify the string that is used to encode and recognize BOOLEAN false values. When an XML document is parsed, the string 0 is always accepted as false for a BOOLEAN element. Enter a string of up to 254 characters.</p> <p>The default is false. 0 is also valid.</p>

## XML representation of null values

Property	Type	Meaning
Encoding Numeric Null	Enumerated type	<p>Specify the null encoding for numeric XML elements. This provides a method of assigning a logical null meaning to such elements. You must select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <code>NULLEmpty</code>. If the element value is the empty string, then the element is null. This is the default value.</li> <li>• <code>NULLValue</code>. If the element value matches that provided by associated property <i>Encoding Numeric Null Value</i>, then the element is null.</li> <li>• <code>NULLXMLSchema</code>. If the element contains an <i>xsi:nil</i> attribute that evaluates to true, then the element is null.</li> <li>• <code>NULLValueAttribute</code>. This option is valid only for elements that have XML Wire Format property <i>Render</i> set to either <code>XMLElementAttrVal</code> or <code>XMLElementAttrIDVal</code>. See “XML Null handling options” on page 676 for details.</li> <li>• <code>NULLAttribute</code> (deprecated). If the element contains an attribute with a name that matches that provided by associated property <i>Encoding Numeric Null Value</i>, and the attribute evaluates to true, then the element is null.</li> <li>• <code>NULLElement</code> (deprecated). If the element contains a child element with a name that matches that provided by associated property <i>Encoding Numeric Null Value</i>, then the element is null.</li> </ul> <p>See “XML Null handling options” on page 676 for full details.</p>
Encoding Numeric Null Value	String	Specify the value to qualify the <i>Encoding Numeric Null</i> property, if you have set that to <code>NULLValue</code> , <code>NULLAttribute</code> , or <code>NULLElement</code> . Refer to “XML Null handling options” on page 676 for further information.
Encoding Non-Numeric Null	Enumerated type	Specify the null encoding for non-numeric XML elements. This provides a method of assigning a logical null meaning to such elements. The options are identical to those available for property <i>Encoding Numeric Null</i> .
Encoding Non-Numeric Null Value	String	Specify the value to qualify the <i>Encoding Non-Numeric Null</i> property. Refer to “XML Null handling options” on page 676 for further information.

## DateTime settings

Property	Type	Meaning
Derive default dateTime format from logical type	Button	<p>Select this option if you want the default dateTime format to be determined by the logical type of the element or attribute. The defaults are:</p> <p><b>dateTime</b>      <code>yyyy-MM-dd'T'HH:mm:ssZZZ</code></p> <p><b>date</b>          <code>yyyy-MM-dd</code></p> <p><b>time</b>          <code>HH:mm:ssZZZ</code></p> <p><b>gYear</b>        <code>yyyy</code></p> <p><b>gYearMonth</b>   <code>yyyy-MM</code></p> <p><b>gMonth</b>        <code>--MM--</code></p> <p><b>gMonthDay</b>    <code>--MM-dd</code></p> <p><b>gDay</b>         <code>---dd</code></p> <p>You can override this property for an element or attribute within a complex type.</p>

Property	Type	Meaning
Use default dateTime Format	Button and dateTime	<p>Select this option if you want to specify a default dateTime format that is fixed for all elements or attributes of logical type dateTime, date, time, gYear, gYearMonth, gMonth, gMonthDay and gDay.</p> <p>The initial value for this property is yyyy-MM-dd'T'HH:mm:ssZZZ, which you can change by over-typing.</p> <p>You can override this property for an element or attribute within a complex type.</p> <p>For more information about dateTime formats, see “DateTime formats” on page 701.</p>
Start of century for 2 digit years	Integer	<p>This property determines how two-digit years are interpreted. Specify the two digits that start a 100-year window that contains the current year. For example, if you specify 89, and the current year is 2002, all two-digit dates are interpreted as being in the range 1989 to 2088.</p> <p>The initial value is 53, which you can change by over-typing.</p>
Days in First Week of Year	Enumerated Type	<p>Specify the number of days of the new year that must fall within the first week.</p> <p>The start of a year usually falls in the middle of a week. If the number of days in that week is less than the value specified here, the week is considered to be the last week of the previous year; hence week 1 starts some days into the new year. Otherwise it is considered to be the first week of the new year; hence week 1 starts some days before the new year.</p> <p>Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a number from the drop-down list. The initial value is 4.</p>
First Day of Week	Enumerated Type	<p>Specify the day on which each new week starts.</p> <p>Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a value from the drop-down list. The initial value is Monday.</p>

Property	Type	Meaning
Strict DateTime Checking	Check box	<p>Select this option if you want to restrict dateTimes to a valid dateTime format. This will not allow 35th March to be processed as 4th April, and 10:79 to be processed as 11:19. If <i>Strict DateTime Checking</i> is set, receiving an incorrect dateTime will cause an error. The default is to restrict dateTimes.</p> <p><b>Strict dateTime checking</b>  Examples of strict dateTime checking are:</p> <ul style="list-style-type: none"> <li>• DateTimes will be restricted to valid dateTimes only. This will not allow dates such as the 35th March to be processed as 4th April, and 10:79 to be processed as 11:19. Receiving an out-of-band dateTime like these examples, will cause an error to occur.</li> <li>• The number of characters for a numeric dateTime component must be within the bounds of the corresponding formatting symbols. Repeat the symbol to specify the minimum number of digits required. The maximum number of digits permitted will be the upper bound for a particular symbol. For example, day in month has an upper bound of 31 therefore a format string of d will allow the values of 2 or 21 to be parsed but will disallow the value of 32 or 210. On output, numbers are padded with zeros to the specified length. A year is a special case, see message set property of Start of century for 2 digit years. For fractional seconds, the length must implicitly match the number of format symbols on input. The output is rounded to the specified length.</li> <li>• Whitespace will not be skipped over. The whitespace in the input string must correspond with the same number and the position of whitespaces in the formatting string.</li> <li>• If there is still data to be parsed in the input string once all the symbols in the formatting string have been matched, an error will occur.</li> </ul> <p><b>Lenient dateTime checking</b>  Examples of lenient dateTime checking are:</p> <ul style="list-style-type: none"> <li>• The parser will convert out-of-band dateTime values to the appropriate in-band value. For example, a date of 2005-05-32 will be converted to 2005-06-01.</li> <li>• Output of dateTimes will always adhere to the symbol count. For example, a formatting string of yyyy-MM-dd (where '-' is the field separator) allows one or more characters to be parsed against MM and dd. This would allow invalid dates such as 2005-1-123 and 2005-011-12 to be input. The first value of 2005-1-123 would be output as the date 2005-05-03, and the second value of 2005-011-12 would be output as the date 2005-11-12.</li> <li>• The number of the timezone formatting symbol Z will only be applicable to the output dateTime format.</li> <li>• Whitespace will be skipped over.</li> </ul>
Time Zone	Enumerated Type	<p>The value that you set for this property is used if the value that you specified for the <i>Default DateTime Format</i> property does not include Time Zone information.</p> <p>The initial value is Use Broker Locale which causes the broker to get the information from the underlying platform.</p> <p>You can change this using the drop down box.</p>



Property	Type	Meaning
Daylight Savings Time	Check box	<p>Select this option if the area in the <i>Time Zone</i> property observes daylight savings time. If it does not observe daylight savings time, this option should not be selected.</p> <p>For example, if an area is selected in <i>Time Zone</i> and this option is not selected, the value passed will represent the time zone without the daylight savings time.</p> <p>Default is not to observe daylight savings time.</p>
Use input UTC format on output	Check box	<p>This property applies to elements and attributes of logical type <code>xsd:dateTime</code> or <code>xsd:time</code> that have a <code>dateTime</code> format of I, IU, T or TU or that includes ZZZ or ZZZU.</p> <p>Such elements and attributes can specify Consolidated Universal Time (UTC) by using either the Z character or <code>timezone +00:00</code> in the value. On input the MRM parser remembers the way that UTC was specified.</p> <p>If this property is checked, and the element or attribute is copied to an output message, then the UTC format will be preserved into the output message and will override that implied by the <code>dateTime</code> format property.</p> <p>If this property is not checked, or the element or attribute was not copied from an input message, then the UTC format in the output message is controlled solely by the <code>dateTime</code> format property.</p> <p>The default for this property is not checked.</p>

### xsi:type output policy

Property	Type	Meaning
Output policy for <code>xsi:type</code> attributes	Enumerated type	<p>When writing XML documents, this property allows you to specify the circumstances under which the <code>xsi:type</code> attribute of elements will be output.</p> <p><b>Never</b> Do not output <code>xsi:type</code> attributes for elements, even if <code>xsi:type</code> attributes appear in the message tree.</p> <p><b>When present</b> Output <code>xsi:type</code> attributes for elements only when <code>xsi:type</code> attributes appear in the message tree.</p> <p><b>Always (Simple elements only)</b> Ensure that all simple elements are output with an <code>xsi:type</code> attribute. If a simple element already has an <code>xsi:type</code> attribute in the message tree it will be used, otherwise an <code>xsi:type</code> attribute will be generated using the rules in the table below.</p> <p><b>Always (All elements)</b> Ensure that all elements are output with an <code>xsi:type</code> attribute if possible to do so. If an element already has an <code>xsi:type</code> attribute in the message tree it will be used, otherwise an <code>xsi:type</code> attribute will be generated using the rules in the table below.</p> <p><b>Follow SOAP Encoding rules</b> Follow the same behaviour as for Always (Simple elements only). Additionally, output a SOAP encoding-style attribute within the root tag of all messages.</p>

If an `xsi:type` attribute needs to be output, but does not appear in the message tree, its value will be generated as follows:

Element type	Value generated when element is defined in model	Value generated when element is self-defining
Simple type	If the type is global or is a built-in type, use it.  If the type is local, use the global or built-in type from which it is derived.	Use the built-in type which best matches the data type of the element in the message tree.
Complex type with simple content	If the type is global use it.  If the type is local, use the global or built-in type from which it is derived.	Use the built-in type which best matches the data type of the element in the message tree.
Complex type with complex content	If the type is global use it.  If the type is local, no xsi:type attribute is output.	No xsi:type attribute is output.

### In-line DTDs and the DOCTYPE text property

You can include in-line DTDs in your messages, and you can specify additional information by setting the property *DOCTYPE Text*, but you must be aware of the action taken by the parser when it constructs an output message:

1. If you take any action that causes the output message to be regenerated, for example if you configure a Compute node to create a new output message by coding ESQL statements like `SET OutputRoot.MRM.Field1 = xxx:`
  - If you have set the property *Suppress DOCTYPE* for the message set in which you have defined this message to Yes, both DOCTYPE information (specified in the *DOCTYPE Text* property for the message set or message) and in-line DTD are excluded from the output message.
  - If you have set the property *Suppress DOCTYPE* for the message set in which you have defined this message to No.
    - The in-line DTD is preserved if possible.
    - Otherwise, if the message is self-defining, the message set *DOCTYPE Text* property information is included in the output message.
    - Otherwise (the message is not self-defining), the message level *DOCTYPE Text* property information is included in the output message.
2. If you do not take any action that causes the output message to be regenerated, the parser generates an output message that is a direct copy of the input message. This occurs if you have configured a Compute node in the message flow to copy the message using `SET OutputRoot = InputRoot` (explicitly, or by checking the *Copy entire message* check box), and you do not modify the message in any way in this or any other node. In this case the in-line DTD is retained in the output message but any information that you specify in the *DOCTYPE Text* property for the message set or message is not included.

### TDS Format message set properties

The tables below shows the message set properties that you can set for the TDS Format. “Default TDS Message set properties” on page 169 shows the defaults for each of the industry standards for each of these properties.

## Messaging Standard

Property	Type	Meaning
Messaging Standard	Enumerated Type	<p>Specify the standard to be used for this wire format. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• ACORD AL3</li> <li>• EDIFACT</li> <li>• SWIFT</li> <li>• UNKNOWN</li> <li>• X12</li> <li>• TLOG</li> <li>• HL7</li> </ul> <p>Select UNKNOWN if you are defining your own tagged/delimited messages, or are using a standard that is not included in the list above.</p> <p>The selected value controls the default values for a number of the other properties.</p> <p>The default is UNKNOWN.</p>

## Data element separation

Property	Type	Meaning
Group Indicator	String	<p>Specify the value of a special character or string that precedes the data belonging to a group or complex type within the bit stream.</p> <p>If you set the group or type property <i>Group Indicator</i>, it overrides this value.</p>
Group Terminator	String	<p>Specify the value of a special character or string that terminates data belonging to a group or a complex type within the bit stream.</p> <p>If you set the group or type property <i>Group Terminator</i>, it overrides this value.</p>
Delimiter	String	<p>Specify the value of a special character or string that specifies the delimiter used between data elements.</p> <p>This property applies only to the delimited <i>Data Element Separation</i> methods (Tagged Delimited, All Elements Delimited, and Variable Elements Delimited).</p>
Suppress Absent Element Delimiters	Enumerated type	<p>Use this property to select if you want delimiters to be suppressed for elements that are missing within a message. Select from:</p> <ul style="list-style-type: none"> <li>• End Of Type. Use this option to suppress the delimiter when an element is missing. For example, if the model has been defined to have up to 3 elements and only 2 are present, the last delimiter can be omitted from the message.</li> <li>• Never. Use this option to ensure that even if optional elements are not present, all delimiters will be written out. This option should be used when the delimiter used to delimit parent and child objects is the same. For example, if an optional child element is missing, message processing applications could not tell where the child elements in a message ended and the next parent element started if the delimiters are all the same.</li> </ul>
Tag Data Separator	Button and String	<p>Specify the value of a special character or string that separates the Tag from the data. The <i>Tag Data Separator</i> and <i>Length of Tag</i> properties are mutually exclusive.</p> <p>If you set the property <i>Tag Data Separator</i>, it overrides <i>Length of Tag</i>.</p> <p>This property applies only to the tagged <i>Data Element Separation</i> methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).</p>

Property	Type	Meaning
Length of Tag	Integer	<p>Specify the length of a tag value. When the message is parsed, this allows tags to be extracted from the bit stream if the <i>Tag Data Separator</i> property is not set.</p> <p>The <i>Tag Data Separator</i> and <i>Length of Tag</i> properties are mutually exclusive. If you set the property <i>Tag Data Separator</i>, it overrides this value.</p> <p>This property applies only to the tagged <i>Data Element Separation</i> methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).</p>

## Character data settings

Property	Type	Meaning
Default CCSID	Integer	<p>CCSID (Coded Character Set Identification) specifies the mapping between character codes and symbols. You must specify a code set that is supported by WebSphere Message Broker.</p> <p>This property stores the default CCSID for the message bit stream, but this value can be overridden when the message is processed (for example by the CCSID in the header of a WebSphere MQ input message).</p>
Trim Fixed Len String	Enumerated Type	<p>Specify if fixed length strings must be trimmed. You can select from:</p> <ul style="list-style-type: none"> <li>No Trim</li> <li>Leading White Spaces</li> <li>Trailing White Spaces</li> <li>Trim Both</li> <li>Trim Padding Chars</li> </ul> <p>The trim of padding characters occurs from the left or right depending on the <i>Justification</i> property for the element.</p> <p>You might need to use this if you have data input that is mapped to a numeric simple type. For example, if the input data has leading spaces, you can set this property to Leading White Spaces to avoid data conversion problems processing these fields.</p>
Escape Character	String	<p>Specify the escape character that is used to allow special reserved characters (such as delimiters) to be included as part of data. You must specify a single character only, or a mnemonic that represents a single character.</p> <ul style="list-style-type: none"> <li>• Escape characters only apply in variable length fields.</li> <li>• Escape characters on parse will always escape the next character and will always be removed. (Variable fields only)</li> <li>• Escape characters on write will be inserted in front of all characters listed in 'Reserved Characters'. (Variable fields only)</li> </ul>
Reserved Characters	String	<p>Specify the special reserved characters that must be preceded by the escape character if they are to be included as part of the data. The Escape Character, delimiters, and group indicators are usually included in this list.</p> <p>If the set of reserved characters is to be updated dynamically (in the case of EDIFACT and X12 when delimiters and so on are specified in service strings), you must use the mnemonics supplied to specify characters in this list.</p> <p>If you have specified reserved characters, an Escape Character must also be specified.</p> <ul style="list-style-type: none"> <li>• Reserved characters only apply in variable length fields.</li> <li>• Reserved Characters are not used on parse.</li> </ul>

## Numeric Settings

Property	Type	Meaning
Decimal Point	String	Specify the character that is used to separate the whole part of a number from its fraction.
Strict Numeric Checking	Boolean	This is used in relation with the <i>Messaging Standard</i> property, <i>Virtual Decimal Point</i> property and the <i>Precision</i> property of a global element. This allows you to apply stricter rules for the checking of numbers. <sup>1</sup>  The default is for this property not to be set.

**Note:**

1. The rules for *Strict Numeric Checking* are:

- If the *Precision* property of a global element is set to All Significant Digits then there should only be a decimal separator if there is a fractional part to the value.
- If the *Precision* property of a global element is set to Explicit Decimal Point then the decimal separator must always be present, even if the fractional part is missing.
- If the *Precision* property of a global element is set to Exponential Notation then the incoming value must be in exponential notation. Exponential notation is only allowed for floating numbers.
- If the *Precision* property of a global element is set to a specific value, then the specific number of digits after the decimal separator must be present.
- All values contain at least one digit in the integer part of the number.
- If a *Virtual Decimal Point* of a global element has been set, the number must not have a decimal point.
- Except for EDIFACT the decimal separator should only be the specified value and '.' should not be permitted. For EDIFACT both '.' and the specified separator are permitted. In this case the decimal separator should be specified as ',' and the code will also permit '.' to be used.
- Except for exponential functions, only digits 0-9, the decimal separator and positive and negative signs are permitted. For exponential functions the characters 'e' and 'E' are also permitted. Padding characters are permitted only if they are in a position to be stripped from the number.

## Representation of boolean values

Property	Type	Meaning
Boolean True Value	String	Specify the value of the string representing the boolean true value. The default value is 1.
Boolean False Value	String	Specify the value of the string representing the boolean false value. The default value is 0.
Boolean Null Value	String	Specify the value of the string representing the boolean null value. The default value is 0.

## DateTime settings

Property	Type	Meaning
Derive default dateTime format from logical type	Button	<p>Select this option if you want the default dateTime format to be determined by the logical type of the element or attribute. The defaults are:</p> <p><b>dateTime</b>      yyyy-MM-dd'T'HH:mm:ssZZZ</p> <p><b>date</b>          yyyy-MM-dd</p> <p><b>time</b>          HH:mm:ssZZZ</p> <p><b>gYear</b>        yyyy</p> <p><b>gYearMonth</b>    yyyy-MM</p> <p><b>gMonth</b>        --MM--</p> <p><b>gMonthDay</b>     --MM-dd</p> <p><b>gDay</b>         ---dd</p> <p>You can override this property for an element or attribute within a complex type.</p>
Use default dateTime Format	Button and dateTime	<p>Select this option if you want to specify a default dateTime format that is fixed for all elements or attributes of logical type dateTime, date, time, gYear, gYearMonth, gMonth, gMonthDay and gDay.</p> <p>The initial value for this property is yyyy-MM-dd'T'HH:mm:ssZZZ, which you can change by over-typing.</p> <p>You can override this property for an element or attribute within a complex type.</p> <p>For more information about dateTime formats, see “DateTime formats” on page 701.</p>
Start of century for 2 digit years	Integer	<p>This property determines how two-digit years are interpreted. Specify the two digits that start a 100-year window that contains the current year. For example, if you specify 89, and the current year is 2002, all two-digit dates are interpreted as being in the range 1989 to 2088.</p> <p>The initial value is 53, which you can change by over-typing.</p>
Days in First Week of Year	Enumerated Type	<p>Specify the number of days of the new year that must fall within the first week.</p> <p>The start of a year usually falls in the middle of a week. If the number of days in that week is less than the value specified here, the week is considered to be the last week of the previous year; hence week 1 starts some days into the new year. Otherwise it is considered to be the first week of the new year; hence week 1 starts some days before the new year.</p> <p>Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a number from the drop-down list. The initial value is 4.</p>
First Day of Week	Enumerated Type	<p>Specify the day on which each new week starts.</p> <p>Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a value from the drop-down list. The initial value is Monday.</p>

Property	Type	Meaning
Strict DateTime Checking	Check box	<p>Select this option if you want to restrict dateTimes to a valid dateTime format. This will not allow 35th March to be processed as 4th April, and 10:79 to be processed as 11:19. If <i>Strict DateTime Checking</i> is set, receiving an incorrect dateTime will cause an error. The default is to restrict dateTimes.</p> <p><b>Strict dateTime checking</b>  Examples of strict dateTime checking are:</p> <ul style="list-style-type: none"> <li>• DateTimes will be restricted to valid dateTimes only. This will not allow dates such as the 35th March to be processed as 4th April, and 10:79 to be processed as 11:19. Receiving an out-of-band dateTime like these examples, will cause an error to occur.</li> <li>• The number of characters for a numeric dateTime component must be within the bounds of the corresponding formatting symbols. Repeat the symbol to specify the minimum number of digits required. The maximum number of digits permitted will be the upper bound for a particular symbol. For example, day in month has an upper bound of 31 therefore a format string of d will allow the values of 2 or 21 to be parsed but will disallow the value of 32 or 210. On output, numbers are padded with zeros to the specified length. A year is a special case, see message set property of Start of century for 2 digit years. For fractional seconds, the length must implicitly match the number of format symbols on input. The output is rounded to the specified length.</li> <li>• Whitespace will not be skipped over. The whitespace in the input string must correspond with the same number and the position of whitespaces in the formatting string.</li> <li>• If there is still data to be parsed in the input string once all the symbols in the formatting string have been matched, an error will occur.</li> </ul> <p><b>Lenient dateTime checking</b>  Examples of lenient dateTime checking are:</p> <ul style="list-style-type: none"> <li>• The parser will convert out-of-band dateTime values to the appropriate in-band value. For example, a date of 2005-05-32 will be converted to 2005-06-01.</li> <li>• Output of dateTimes will always adhere to the symbol count. For example, a formatting string of yyyy-MM-dd (where '-' is the field separator) allows one or more characters to be parsed against MM and dd. This would allow invalid dates such as 2005-1-123 and 2005-011-12 to be input. The first value of 2005-1-123 would be output as the date 2005-05-03, and the second value of 2005-011-12 would be output as the date 2005-11-12.</li> <li>• The number of the timezone formatting symbol Z will only be applicable to the output dateTime format.</li> <li>• Whitespace will be skipped over.</li> </ul>
Time Zone	Enumerated Type	<p>The value that you set for this property is used if the value that you specified for the <i>Default DateTime Format</i> property does not include Time Zone information.</p> <p>The initial value is Use Broker Locale which causes the broker to get the information from the underlying platform.</p> <p>You can change this using the drop down box.</p>

Property	Type	Meaning
Daylight Savings Time	Check box	<p>Select this option if the area in the <i>Time Zone</i> property observes daylight savings time. If it does not observe daylight savings time, this option should not be selected.</p> <p>For example, if an area is selected in <i>Time Zone</i> and this option is not selected, the value passed will represent the time zone without the daylight savings time.</p> <p>Default is not to observe daylight savings time.</p>
Use input UTC format on output	Check box	<p>This property applies to elements and attributes of logical type <code>xsd:dateTime</code> or <code>xsd:time</code> that have a <code>dateTime</code> format of I, IU, T or TU or that includes ZZZ or ZZZU.</p> <p>Such elements and attributes can specify Consolidated Universal Time (UTC) by using either the Z character or <code>timezone +00:00</code> in the value. On input the MRM parser remembers the way that UTC was specified.</p> <p>If this property is checked, and the element or attribute is copied to an output message, then the UTC format will be preserved into the output message and will override that implied by the <code>dateTime</code> format property.</p> <p>If this property is not checked, or the element or attribute was not copied from an input message, then the UTC format in the output message is controlled solely by the <code>dateTime</code> format property.</p> <p>The default for this property is not checked.</p>

## TDS Mnemonics

The Tagged/Delimited String Format (TDS) uses mnemonics for a number of properties for a message set, complex type, or both. These TDS mnemonics and their associated properties are listed in the table below.

Mnemonic string	Meaning	Default value	Associated property
<EDIFACT_CS>	Component separator in EDIFACT	:	Message set and complex type/group, <i>Delimiter</i>
<EDIFACT_DS>	Data element separator in EDIFACT	+	Message set and complex type/group, <i>Delimiter</i>
<EDIFACT_TAGDATA_SEP>	Tag data separator in EDIFACT  This is overridden with the same value as that which overrides <EDIFACT_DS>	+	Message set and complex type/group, <i>Tag Data Separator</i>
<EDIFACT_DEC_NOTATION>	Decimal notation in EDIFACT	.	Message set, <i>Decimal Point</i>
<EDIFACT_ESC_CHAR>	Escape character in EDIFACT	?	Message set, <i>Escape Character</i>
<EDIFACT_GROUP_TERM>	Tag terminator in EDIFACT	'	Message set, <i>Group Terminator</i>
<X12_GROUP_TERM>	Tag terminator in X12	!	Message set level, <i>Group Terminator</i>
<X12_DS>	Data element separator for X12	*	Message set and complex type/group, <i>Delimiter</i>
<X12_CS>	Component separator for X12	:	Message set and complex type/group, <i>Delimiter</i>
<HL7_CS>	Component separator in HL7	^	Message set and complex type/group, <i>Delimiter</i>



Mnemonic string	Meaning	Default value	Associated property
<HL7_FS>	Data element separator in HL7		Message set and complex type/group, <i>Delimiter</i>
<HL7_RS>	Repeating element delimiter in HL7	~	Local element and element reference, <i>Repeating Element Delimiter</i>
<HL7_SCS>	Sub-component separator in HL7	&	Message set and complex type/group, <i>Delimiter</i>

Mnemonics are also supported for the following control characters:

<ACK> (x'06')	<BEL> (x'07')	<BS> (x'08')	<CAN> (x'18')
<CR> (x'0D')	<DC1> (x'11')	<DC2> (x'12')	<DC3> (x'13')
<DC4> (x'14')	<DLE> (x'10')	<EM> (x'19')	<ENQ> (x'05')
<EOT> (x'04')	<ESC> (x'1B')	<ETB> (x'17')	<ETX> (x'03')
<FF> (x'0C')	<FS> (x'1C')	<GS> (x'1D')	<GT> (x'3E')
<HT> (x'09')	<LF> (x'0A')	<LT> (x'3C')	<NAK> (x'15')
<NUL> (x'00')	<RS> (x'1E')	<SI> (x'0F')	<SO> (x'0E')
<SOH> (x'01')	<SP> (x'20')	<STX> (x'02')	<SUB> (x'1A')
<SYN> (x'16')	<US> (x'1F')	<VT> (x'0B')	

You can enter a mnemonic in the form <U+xxxx> where xxxx are hexadecimal numbers up to a value of FFFF. These numbers represent a Unicode character, not a character in your local code page or the code page in which message data is formatted. None of the characters in this structure are case sensitive. Do not enclose spaces inside the angle brackets.

## Default TDS Message set properties

The following tables define the defaults for the message set properties for the TDS Format for each of the industry standard messages that you can define. For more information about the TDS Format, see “TDS Format message set properties” on page 162 and “TDS Mnemonics” on page 168.

### Default message set property values for TDS (part 1 of 2)

Property	Messaging standard = EDIFACT	Messaging Standard = HL7	Messaging standard = SWIFT
Group Indicator	Empty	Empty	<CR><LF>:
Group Terminator	<EDIFACT_GROUP_TERM>	<CR>	<CR><LF>-
Delimiter	<EDIFACT_CS>	<HL7_FS>	<CR><LF>:
Suppress Absent Element Delimiters	End of Type	End of Type	End of Type
Tag Data Separator	<EDIFACT_TAGDATA_SEP>	<HL7_FS>	:
Length of Tag	Empty	Empty	Empty
Default CCSID	367	367	37
Trim Fix Len String	No Trim	No Trim	No Trim
Escape Character	<EDIFACT_ESC_CHAR>	Empty	Empty

Property	Messaging standard = EDIFACT	Messaging Standard = HL7	Messaging standard = SWIFT
Reserved Chars	<EDIFACT_ESC_CHAR> <EDIFACT_TAGDATA_SEP> <EDIFACT_GROUP_TERM> <EDIFACT_CS>	Empty	Empty
Decimal Point	<EDIFACT_DEC_NOTATION>	.	,
Strict Numeric Checking	Checked	Unchecked	Checked
Boolean True Value	1	1	1
Boolean False Value	0	0	0
Boolean Null Value	0	0	0
Default DateTime Format <sup>1</sup>	Derive default dateTime format from logical type	yyyyMMddHHmmss	Derive default dateTime format from logical type
Start of century for 2 digit years	53	53	80
Days in First Week of Year	Use Broker Locale	Use Broker Locale	Use Broker Locale
First Day of Week	Use Broker Locale	Use Broker Locale	Use Broker Locale
Strict Datetime Checking	Unchecked	Unchecked	Unchecked
Time Zone	Use Broker Locale	Use Broker Locale	Use Broker Locale
Daylight Savings Time	Unchecked	Unchecked	Unchecked
<b>Note:</b>			
1. The Default DateTime Format property should be activated by additionally selecting the Use Default DateTime Format radio button.			

### Default message set property values for TDS (part 2 of 2)

Property	Messaging standard = UNKNOWN	Messaging standard = X12	Messaging standard = ACORD AL3
Group Indicator	Empty	Empty	Empty
Group Terminator	Empty	<X12_GROUP_TERM>	Empty
Delimiter	Empty	<X12_CS>	Empty
Suppress Absent Element Delimiters	End of Type	End of Type	End of Type
Tag Data Separator	Empty	Empty	Empty
Length of Tag	Empty	Empty	Empty
Default CCSID	376	367	367
Trim Fix Len String	No Trim	Trim Both	No Trim
Escape Character	Empty	Empty	Empty
Reserved Chars	Empty	Empty	Empty
Decimal Point	.	.	.
Strict Numeric Checking	Unchecked	Checked	Checked
Boolean True Value	1	1	Y

Property	Messaging standard = UNKNOWN	Messaging standard = X12	Messaging standard = ACORD AL3
Boolean False Value	0	0	N
Boolean Null Value	0	0	N
Default DateTime Format <sup>1</sup>	Derive default dateTime format from logical type	Derive default dateTime format from logical type	Derive default dateTime format from logical type
Start of century for 2 digit years	53	53	53
Days in First Week of Year	Use Broker Locale	Use Broker Locale	Use Broker Locale
First Day of Week	Use Broker Locale	Use Broker Locale	Use Broker Locale
Strict Datetime Checking	Unchecked	Unchecked	Unchecked
Time Zone	Use Broker Locale	Use Broker Locale	Use Broker Locale
Daylight Savings Time	Unchecked	Unchecked	Unchecked
<b>Note:</b>			
1. The Default DateTime Format property should be activated by additionally selecting the Use Default DateTime Format radio button.			

### Default complex type/group property values for TDS (part 1 of 2)

Property	Messaging standard = EDIFACT	Messaging standard = HL7	Messaging standard = SWIFT
Data Element Separation	All Elements Delimited	All Elements Delimited	Tagged Delimited
Group Indicator	Empty	Empty	<CR><LF>:
Group Terminator	<EDIFACT_GROUP_TERM>	<CR>	<CR><LF>-
Delimiter	<EDIFACT_CS>	<HL7_FS>	<CR><LF>:
Suppress Absent Element Delimiters	End of Type	End of Type	End of Type
Observe Element Length	Unchecked	Unchecked	Unchecked
Tag Data Separator	<EDIFACT_DATA_SEP>	<HL7_FS>	:
Length of Tag	Empty	Empty	Empty
Length of Encoded Length	not applicable	not applicable	not applicable
Extra Chars in Encoded Length	not applicable	not applicable	not applicable

### Default complex type/group property values for TDS (part 2 of 2)

Property	Messaging standard = UNKNOWN	Messaging standard = X12	Messaging standard = ACORD AL3
Data Element Separation	Fixed Length	All Elements Delimited	Fixed Length AL3
Group Indicator	Empty	Empty	Empty

Property	Messaging standard = UNKNOWN	Messaging standard = X12	Messaging standard = ACORD AL3
Group Terminator	Empty	<X12_GROUP_TERM>	Empty
Delimiter	Empty	<X12_CS>	not applicable
Suppress Absent Element Delimiters	End of Type	End of Type	End of Type
Observe Element Length	Checked	Unchecked	Checked
Tag Data Separator	Empty	Empty	Empty
Length of Tag	Empty	Empty	Empty
Length of Encoded Length	not applicable	not applicable	not applicable
Extra Chars in Encoded Length	not applicable	not applicable	not applicable

## Documentation properties for a message set

Property	Type	Meaning
Version	String	This field allows you to enter a version for the message set. This allows the version of the message set to be displayed using the Eclipse properties view.  A default for this field can be set in the messages set preferences.
Documentation	String	The documentation property of a message set is where you can add information to enhance the understanding of the message set's function.  It is a string field and any standard alphanumeric characters can be used.  You can also use this field to define a keyword and its value that will display for the deployed message set in the properties view of Eclipse. An example is: \$MQSI Author=Fred MQSI\$  When the properties of the deployed message set are displayed, this will add a row to the display showing Author as the property name and 'Fred' as its value.

## Guidance for defining keywords

This topic contains the rules to follow when defining keywords. Keywords and their values are displayed in the properties view of a deployed object.

A number of objects in WebSphere Message Broker can have additional information added to the object. This information can display information about an object after the object has been deployed. The default information that is displayed is the time the object was deployed and the last time the object was modified. Version information can be specifically set using the **Version** property for message sets and message flows.

You can define custom keywords and their values that the Configuration Manager will interpret as additional information to be displayed in the properties view. For example, you can define keywords for "Author" and "Subflow 1 Version":

```
$MQSI Author=John Smith MQSI$
$MQSI Subflow 1 Version=v1.3.2 MQSI$
```

The information the Configuration Manager shows is:

Object name	
Deployment Time	28-Aug-2004 15:04
Modification Time	28-Aug-2004 14:27
Version	v1.0
Author	John Smith
Subflow 1 Version	v1.3.2

In this display the version information has also been defined using the Version property of the object. If the version information had not been defined using the property, it would be omitted from this display.

The syntax for defining a keyword and its associated value is:

`$MQSI KeywordName = KeywordValue MQSI$`

Where:

### **\$MQSI**

\$MQSI opens the definition. It can be followed by an optional underscore or white space character that is ignored.

### *KeywordName*

The name of the keyword that you are setting the value for. It can be made up of any sequence of alphanumeric characters apart from the equals (=) sign. It can contain white space characters, but any leading or trailing white space characters will be omitted.

= The equals (=) sign is the delimiter between the keyword and the value that you are setting it to.

### *KeywordValue*

The value that the keyword will be set to. It can be made up of any sequence of alphanumeric characters. It can contain white space characters, but any leading or trailing white space characters will be omitted.

### **MQSI\$**

MQSI\$ closes the keyword definition.

## **Examples**

Example definitions	Interpreted keyword and value	Comments
\$MQSIAuthor=JohnMQSI\$ or \$MQSI Author=John MQSI\$ or \$MQSI Author = John MQSI\$	Keyword = "Author" Value = "John"	Each of these is a basic example of what can be set and shows that the leading and trailing white space characters for the name and value parameters is ignored.
\$MQSI_Author = John MQSI\$	Keyword = "Author" Value = "John"	The first character after \$MQSI can be an underscore character. The underscore character is omitted in the interpreted keyword. If a second underscore character appears, this will form part of the keyword name.
\$MQSI Flow designer = John Smith MQSI\$	Keyword = "Flow designer" Value = "John Smith"	White space characters are accepted for each parameter value.

Example definitions	Interpreted keyword and value	Comments
\$MQSI bar = MQSI\$	Keyword = "bar" Value = ""	The keyword value can be set to an empty ("" ) string.
\$MQSI_mqsitag=\$MQSI\$MQSI\$	Keyword = "mqsitag" Value = "\$"	This is a poorly formatted definition. After defining the keyword name the parser is looking to find the delimiters that form the boundary of the value to be set. In this case the only character prior to the MQSI\$ that closes the definition is a '\$' and that is set as the keyword value.
\$MQSI=barMQSI\$		This pattern is ignored because the keyword name cannot be an empty string.
\$MQSItagbarMQSI\$		This pattern is ignored because there is not a separator (=) between the keyword name and the keyword value.

## Message definition file properties

### Namespace

Property	Type	Meaning
Prefix	String	The namespace prefix for the target namespace of this file. This field cannot be changed after the message definition file has been created.
Target Namespace	String	The target namespace for the message definition file. All global objects created within the file will have this namespace by default. This field cannot be changed after the message definition file has been created.

### Default namespaces for local objects

Property	Type	Meaning
Elements	String	The default namespace for all local elements within this message definition file.
Attributes	String	The default namespace for all local attributes within this message definition file.

Property	Type	Meaning
Default block	String	The default value for the block attribute for all complex types and elements within this message definition file.
Default final		The default value for the final attribute for all complex types and elements within this message definition file.

Property	Type	Meaning
Use <a href="http://www.w3.org/2001/xml.xsd">http://www.w3.org/2001/xml.xsd</a> schema	Check box	Set this property if you need to use the <a href="http://www.w3.org/2001/xml.xsd">http://www.w3.org/2001/xml.xsd</a> schema. When you set this property it imports the <a href="http://www.w3.org/2001/xml.xsd">http://www.w3.org/2001/xml.xsd</a> schema and allows you to use any of the constructs in that schema.

## Message definition file includes properties

Property	Type	Meaning
Schema Location	String	For each message definition file that has been included in this message definition file, this field displays its location. The location is displayed as a relative path from the message definition file to the included file.

## Message definition file imports properties

Property	Type	Meaning
Schema Location	String	For each message definition file that has been imported into this message definition file, this field displays its location. The location is displayed as a relative path from the message definition file to the imported file.
Prefix	String	Displays the namespace prefix for each imported message definition file.
Namespace	String	Displays the namespace URI for each imported message definition file.

## Message definition file redefines properties

This provides details of the properties associated with message definition redefines.

Property	Type	Meaning
Schema Location	String	For each message definition file that has been redefined in this message definition file, this field displays its location. The location is displayed as a relative path from the message definition file to the included file.

**Note:** Redefines are not supported and will result in a validation error. If you right-click on the error message and select **Quick Fix** you can choose to convert the redefines construct into an include construct. This also removes the error message.

## Documentation properties for all message set objects

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

**Note:**

1. Key, Keyref, and Unique objects do not have documentation properties.

---

## Message category properties

This describes the properties associated with a message category.

Property	Type	Meaning
Message Category Kind	Enumerated type	<p>This property describes the purpose of the message category. Choose from:</p> <ul style="list-style-type: none"> <li>• wsdl. This is the default. If the message category is to participate in the generation of WSDL documents, this property must be set to 'wsdl'. When the WSDL document is generated, the name of the message category provides the name for the &lt;wsdl:operation&gt; element generated for eligible messages in the message category.</li> <li>• other. Indicates the category represents a generic grouping of messages as an aid to organising them in your workspace.</li> </ul>
Message Category Usage	Enumerated type	<p>This property describes the operation type for a WSDL operation. Choose from:</p> <ul style="list-style-type: none"> <li>• wsdl:request-response. This is the default for a Message Category of kind wsdl.</li> <li>• wsdl:solicit-response...</li> <li>• wsdl:one-way...</li> <li>• wsdl:notification...</li> <li>• empty string. This is the default for a Message Category of kind other.</li> </ul>
Documentation	String	<p>The documentation property of an object is where you can add information to enhance the understanding of that objects function.</p> <p>It is a string field and any standard alphanumeric characters can be used.</p> <p>For a Message Category of kind wsdl the value of the field is included in any generated WSDL as the wsdl:documentation child of operation element in the WSDL portType.</p> <p>For a Message Category of kind other the value serves only to document the Message Category within your workspace.</p>

## Message category member properties

This describes the properties associated with a message category member.

Property	Type	Meaning
Role Name	String	<p>For a Message Category of kind wsdl the value of the field becomes the WSDL message part name and must be unique within the category. It defaults to the message name.</p> <p>For a Message Category of kind other the value of the field has no particular significance and defaults to an empty string.</p>
Role Type	Enumerated type	<p>The role that the message plays in the message category. Select from:</p> <ul style="list-style-type: none"> <li>• wsdl:input</li> <li>• wsdl:output</li> <li>• wsdl:return</li> <li>• wsdl:fault</li> <li>• empty string</li> </ul> <p>For a Message Category of kind wsdl the default is wsdl:input. This field dictates the role within a WSDL operation. The value wsdl:return implies wsdl:output, but for rpc-style WSDL generation it also identifies the message part used as the return value and in this instance can be omitted from the parameterOrder attribute. At most one message may have Role Type of wsdl:return.</p> <p>For a Message Category of kind other the value defaults to an empty string and this property has no role in the message category.</p>



Property	Type	Meaning
Role Usage	Enumerated type	<p>The role that the message plays in the SOAP binding. Select from:</p> <ul style="list-style-type: none"> <li>• soap:body</li> <li>• soap:header</li> <li>• soap:fault</li> <li>• soap:headerfault</li> <li>• empty string</li> </ul> <p>For a Message Category of kind wsdl this field defaults to soap:body and dictates the SOAP-binding child of the WSDL input, output or fault element.</p> <p>For a Message Category of kind other this field is deactivated.</p>
Documentation	String	<p>It is a string field and any standard alphanumeric characters can be used.</p> <p>For a Message Category of kind wsdl the value of the field is included in any generated WSDL as the wsdl:documentation child of the WSDL input, output or fault element under the WSDL portType.</p> <p>For a Message Category of kind other the value serves only to document the Message Category within your workspace.</p>

---

## Message model object properties

There are two ways of accessing the reference information for the properties of message model objects. The following topics allow you to access the property information by property kind:

- “Logical properties for message model objects”
- “Physical properties for message model objects” on page 203
- “Documentation properties for all message set objects” on page 175

Alternatively, you can access the property information by object, starting from:

- “Message model object properties by object” on page 233

Deprecated objects are dealt with separately. For further information, see “Deprecated message model object properties” on page 549

## Logical properties for message model objects

Logical property information is available for the following objects:

- “Attribute group reference logical properties” on page 178
- “Attribute reference logical properties” on page 178
- “Complex type logical properties” on page 179
- “Element reference logical properties” on page 183
- “Global attribute logical properties” on page 184
- “Global attribute group logical properties” on page 187
- “Global element logical properties” on page 187
- “Global group logical properties” on page 190
- “Group reference logical properties” on page 191
- “Key logical properties” on page 192
- “Keyref logical properties” on page 192

- “Local element logical properties” on page 195
- “Local group logical properties” on page 198
- “Message logical properties” on page 200
- “Simple type logical properties” on page 201
- “Unique logical properties” on page 202
- “Wildcard attribute logical properties” on page 202
- “Wildcard element logical properties” on page 202

### Attribute group reference logical properties

Property	Type	Meaning
Reference Name	Enumerated type	The <i>Reference Name</i> is the name of the object that this object is referring to. The objects available to reference can be selected from the drop down list.

### Attribute reference logical properties

Property	Type	Meaning
Reference Name	Enumerated type	The <i>Reference Name</i> is the name of the object that this object is referring to. The objects available to reference can be selected from the drop down list.

Property	Type	Meaning
Usage	Enumerated type	<p>The usage property is used in conjunction with the <i>Value</i> property found in an attribute object. The default for the <i>Usage</i> property is <i>optional</i>.</p> <p>Select from;</p> <ul style="list-style-type: none"> <li>• <i>optional</i>. <ul style="list-style-type: none"> <li>– Where the <i>Value</i> property is set to default and no data has been entered in the <i>Value</i> property, the attribute can appear once and can have any value.</li> <li>– Where the <i>Value</i> property is set to default, the attribute can appear once. If it does not appear, its value will be the data that has been entered in the <i>Value</i> property. If it does appear it will be the value given.</li> <li>– Where the <i>Value</i> property is set to fixed, the attribute can appear once. If it does appear, its value must match the data that has been entered in the <i>Value</i> property. If it does not appear its value will be the data that has been entered in the <i>Value</i> property.</li> </ul> </li> <li>• <i>prohibited</i>. The attribute must not appear.</li> <li>• <i>required</i>. <ul style="list-style-type: none"> <li>– Where the <i>Value</i> property is set to default and no data has been entered in the <i>Value</i> property, the attribute must appear once and can have any value.</li> <li>– Where the <i>Value</i> property is set to fixed, the attribute must appear once and it must match the data that has been entered in the <i>Value</i> property.</li> </ul> </li> </ul>

## Complex type logical properties

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"> <li>• - the hyphen</li> <li>• _ the underscore</li> <li>• . the period</li> </ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <i>xml</i> or any variant (for example <i>Xml</i>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>
Base Type	Enumerated type	You can use this property to select a type (simple or complex) that is used as the starting point to define a new complex type that is derived by restriction or extension.
Derived By	Enumerated type	<p>If this property is active, select from;</p> <ul style="list-style-type: none"> <li>• restriction. If a complex type is derived by restriction, the content model of the complex type is a subset of the base type.</li> <li>• extension. If the complex type is derived by extension, the content model of the complex type is the content model of the base type plus the content model specified in the type derivation.</li> </ul> <p>Derivation by list or union is not supported.</p>

## Content

The table below shows the valid settings for *Composition* and *Content Validation*. These properties are actually located on the group which defines the content of this type. They can only be edited if the *Local group* button is selected. If the *Global group* button is selected, these properties are taken from the global group identified by the *Group name* field.

Valid children in a complex type that depend on both *Composition* and *Content Validation* are shown in "Content Validation properties for complex types" on page 182.

Property	Type	Meaning
Local Group	Button	You should select this if the content of your complex type is a local group.

Property	Type	Meaning
Composition	Enumerated type	<p>The property describes how the message tree is structured and is used in combination with the property <i>Content Validation</i>.</p> <p><i>Composition</i> determines, for example, if the elements within the tree can appear in any order, or if the order is predefined.</p> <p>If you set this property to Ordered Set or Sequence, the order of elements in the input message when the message is parsed, and the order in the logical tree when the output message is constructed by the parser, is important. If the order is not correct, the parser might generate an error, or might produce unexpected results. Therefore you must take care to include ESQL SET statements in the correct order when you create a message in a Compute node.</p> <p>Select from:</p> <ul style="list-style-type: none"> <li>• Empty</li> <li>• sequence. If you select this option, you can only define children that are simple types, complex types, elements, or groups. These children, if present, must appear in the specified order. They can repeat and can be duplicated.</li> <li>• choice. If you select this option, you can define children that are simple types, complex types, or elements. Only one of the defined children of the complex type can be present, but repeating children are allowed. Use this option if you want to model C unions and COBOL REDEFINES in a Custom Wire Format, or an XML DTD element that uses choice in an XML Wire Format, or some industry standard tagged/delimited messages (for example SWIFT) use this format.</li> <li>• all. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can only contain elements - groups are not allowed. An all group can only be used at the top level of a complex type - it cannot be a member of another group within a type.</li> <li>• unorderedSet. If you select this option, you can define only elements as children. The elements can repeat but cannot be duplicated. Child elements can appear in any order.</li> <li>• orderedSet. If you select this option, you can define only elements as children. These elements, if present, must appear in the specified order, and they can repeat but cannot be duplicated. This is the default value for new complex types.</li> <li>• message. If you select this option, you can define only messages as children. They can repeat, but they cannot be duplicated. Like Choice, only one of the defined children can be present. If the complex type includes more than one message, the bit stream contains the exact length of the embedded message, and is not padded to the length of the longest. Use this option to model multipart messages, which are used in some industry standards, for example, SWIFT. For more information, see the section on multipart messages in "Multipart messages" on page 28.</li> </ul>

Property	Type	Meaning
Content Validation	Enumerated type	<p><i>Content Validation</i> controls how the broker responds to undeclared content and specifies where the objects that are included within the complex type are defined, if at all. It is used in combination with the <i>Composition</i> property.</p> <p>Options:</p> <ul style="list-style-type: none"> <li>• <i>Closed</i>. The complex type can only contain the child elements that you have added to it.</li> <li>• <i>Open Defined</i>. The complex type can contain any valid element defined within the message set.</li> <li>• <i>Open</i>. The complex type can contain any valid element, not just those that you have added to this complex type.</li> </ul> <p>See “Combinations of Composition and Content Validation” on page 280 for further details of these options.</p>
Group Reference	Button	You should select this if the content of your complex type is a reference to a group object
Group Name	Enumerated type	The <i>Group Name</i> is the name of the group that this complex type is referring to. The groups available to be referenced can be selected from the drop down list.
Min Occurs	Integer	<p>Specify the minimum number of times that the object can repeat. The default is 1.</p> <p>If the value is set to 0, then the object is optional.</p> <p>With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i>, it must be less than or equal to the value in <i>Max Occurs</i>.</p>
Max Occurs	Integer	<p>Specify the maximum number of times that the object can repeat. The default is 1.</p> <p>If this property is not set, then the object can not occur more than once.</p> <p>If this property is set to 0, it will be interpreted as if the object does not exist in the message.</p> <p>It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.</p>
Mixed	Check box	Select this where the complex type has mixed content and contains character data alongside sub-elements.

### Substitution settings

Property	Type	Meaning
Final	Multiple selection enumerated type	<p>The final attribute on a complex type controls whether other types may be derived from it. Valid values are extension/restriction/all. You can select from one or more of the following:</p> <ul style="list-style-type: none"> <li>• <i>Empty</i></li> <li>• <i>restriction</i>. Prohibit type substitution by elements whose types are restrictions of the head element’s type.</li> <li>• <i>extension</i>. Prohibit type substitution by elements whose types are extensions of the head element’s type.</li> <li>• <i>all</i>. Prohibit substitution by any method.</li> </ul> <p>To select more than one, you will need to type the selection into the property field.</p>

Property	Type	Meaning
Block	Multiple selection enumerated type	<p>The block attribute on a complex type restricts the types of substitutions which are allowed for elements based on that type. In the WebSphere Message Broker its effect is the same as if the block attribute were copied from the complex type onto every element based on the complex type. You can select from one or more of the following:</p> <ul style="list-style-type: none"> <li>• Empty</li> <li>• restriction. Prohibit type substitution by elements whose types are restrictions of the head element's type.</li> <li>• extension. Prohibit type substitution by elements whose types are extensions of the head element's type.</li> <li>• all. Prohibit substitution by any method.</li> </ul> <p>To select more than one, you will need to type the selection into the property field.</p>
Abstract	Check box	If selected, no elements based on this type can appear in the message.

### Content Validation properties for complex types:

*Content Validation* specifies where the objects that are included within a complex type or group are defined, if at all. It is used in combination with *Composition*.

The first table below shows the valid settings for *Content Validation* if *Composition* is set to Message, and the second table shows the valid settings for *Content Validation* if *Composition* is not set to Message.

### Content Validation options if Composition is set to Message

Option	Meaning
Open	When a message is parsed, this complex type or group can contain any message, not just those that you have defined in this message set. You can use this option for sparse messages (see "Predefined and self-defining elements and messages" on page 32 for a definition of sparse messages).
Closed	When a message is parsed, this complex type or group can only contain the messages that are members of this complex type or group. This is always the case for messages represented in CWF format.
Open Defined	When a message is parsed, this complex type or group can contain any message defined within the message set.

### Content Validation options if Composition is not set to Message

Option	Meaning
Open	When a message is parsed, this complex type or group can contain any elements and not just those that you have defined in this message set (see "Predefined and self-defining elements and messages" on page 32 for a definition of sparse messages).
Closed	When a message is parsed, this complex type or group can only contain the elements that are members of this complex type or group.
Open Defined	When a message is parsed, this complex type or group can contain any element that you have defined within the message set.

When you are using Content Validation set to open or open defined, you can not specify the precise position where the content that is not modeled is permitted to

occur. If you wish to do this, you should consider using a wildcard element as an alternative. Note that wildcard elements can only appear within a complex type or group with Composition of sequence and Content Validation of closed.

### Element reference logical properties

Property	Type	Meaning
Reference Name	Enumerated type	The <i>Reference Name</i> is the name of the object that this object is referring to. The objects available to reference can be selected from the drop down list.

### Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default is 1.  If the value is set to 0, then the object is optional.  With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i> , it must be less than or equal to the value in <i>Max Occurs</i> .
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default is 1.  If this property is not set, then the object can not occur more than once.  If this property is set to 0, it will be interpreted as if the object does not exist in the message.  It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.

The Min Occurs and Max Occurs properties are used in conjunction with an elements Value properties. The table below summarizes how an element reference can be constrained.

Min Occurs	Max Occurs	Fixed	Default	Notes
1	1			The element must appear once and can have any value.
1	1	Delta		The element must appear once and it must match the data that has been entered in the <i>Value</i> property. In this example the element must contain the text Del ta.
2	-1	Delta		The element must appear twice or more and it must match the data that has been entered in the <i>Value</i> property. In this example there will be at least two elements that must contain the text Del ta.
0	1			The element is optional and can appear once and have any value.
0	1	Delta		The element is optional and can appear once. If it does appear, its value must match the data that has been entered in the <i>Value</i> property. If it does not appear its value will be the data that has been entered in the <i>Value</i> property.

Min Occurs	Max Occurs	Fixed	Default	Notes
0	1		Delta	The element is optional and can appear once. If it does not appear, its value will be the data that has been entered in the <i>Value</i> property. If it does appear it must be the value given in the element.
0	2		Delta	The element is optional and can appear once, twice or not at all. If the element does not appear it is not provided. If the element appears and it is empty, it set to the data held in the <i>Value</i> property, else it is the value given in the element.
0	0			The element is prohibited and must not appear.

### Global attribute logical properties

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"> <li>• - the hyphen</li> <li>• _ the underscore</li> <li>• . the period</li> </ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <i>xml</i> or any variant (for example <i>Xml</i>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>



Property	Type	Meaning
Type	Enumerated type	<p>The Type property constrains the type of data that can be present in the object.</p> <p>There are a limited number of types available directly from the drop down selector. These are;</p> <ul style="list-style-type: none"> <li>• int</li> <li>• string</li> <li>• boolean</li> <li>• hexBinary</li> <li>• dateTime</li> <li>• date</li> <li>• time</li> <li>• decimal</li> <li>• float</li> <li>• (More...)</li> <li>• (New Simple Type)</li> <li>• (New Complex Type)</li> </ul> <p>If you select (More...), this starts the Type Selection wizard. From this wizard you can select any of the available types.</p> <p>If you select (New Simple Type), this starts the New Simple Type wizard which allows you to create an Anonymous simple type which will be based on an existing type. This can be created locally or globally.</p> <p>If you select (New Complex Type), this starts the New Complex Type wizard which allows you to create an Anonymous complex type which can be derived from an existing base type. This can be created locally or globally.</p> <p>For further information about these types, and examples of their use see the XML Schema Part 0: Primer which can be found on the World Wide Web Consortium (W3C) Web site.</p>
Namespace	Enumerated type	<p>Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references.</p> <p>If &lt;no target namespace&gt; is displayed, a namespace has not been set for this object.</p> <p>If the property is inactive, the message set has not been configured to support namespaces.</p> <p>Where the property is active, namespaces that are available for selection will be displayed in the drop down list.</p>

## Value

The *Value* properties are used in conjunction with the *Usage* property in an Attribute Reference or a Local Attribute.

Property	Type	Meaning
Default	Button and String	<p>This property provides the default value for the element or attribute.</p> <p>When writing a fixed-length portion of a message (CWF or fixed-length TDS) if the attribute or element is missing from the message tree, the default value will be inserted into the bit stream in order to preserve the message structure.</p> <p>This property is not used by the message broker when parsing.</p> <p>Note that this behavior differs from the XML Schema specification.</p>
Fixed	Button and String	<p>If a <i>Fixed</i> value is set for an attribute and a message is received, if the attribute exists in the message, the data in the attribute property of the message must match that held in the fixed value of the attribute definition.</p> <p>If the attribute does not exist in the message, the broker will create the attribute and populate it with the data held in the fixed value of the attribute definition.</p>
Interpret Value As	Enumerated Type	<p>Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur.</p> <p>The possible values for this property are:</p> <ul style="list-style-type: none"> <li>• None This is the default value and will indicate that the element or attribute does not have any key value associated with it.</li> <li>• Message Set Identity. Specifies that the value of the element or attribute will correspond to the identifier, name or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value will remain in force unless a new element or attribute Message Set Identity field is encountered which resets the Message Set Identity value to this new one.</li> <li>• Message Identity. Specifies that the value of the element or attribute will correspond to the name or alias (in that priority order) that is associated with a message and will act as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity will apply until a new element or attribute Message Identity field is encountered at the same level in the tree. The embedded message may be defined in either the current message set or in a message set identified using a Message Set Identity.</li> </ul> <p><b>Note:</b> This property is only applicable when the type of the object is derived from <code>xsd:string</code>.</p>

## Global attribute group logical properties

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"> <li>• - the hyphen</li> <li>• _ the underscore</li> <li>• . the period</li> </ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <code>xml</code> or any variant (for example <code>Xml</code>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>

## Global element logical properties

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"> <li>• - the hyphen</li> <li>• _ the underscore</li> <li>• . the period</li> </ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <code>xml</code> or any variant (for example <code>Xml</code>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>

Property	Type	Meaning
Type	Enumerated type	<p>The Type property constrains the type of data that can be present in the object.</p> <p>There are a limited number of types available directly from the drop down selector. These are;</p> <ul style="list-style-type: none"> <li>• int</li> <li>• string</li> <li>• boolean</li> <li>• hexBinary</li> <li>• dateTime</li> <li>• date</li> <li>• time</li> <li>• decimal</li> <li>• float</li> <li>• (More...)</li> <li>• (New Simple Type)</li> <li>• (New Complex Type)</li> </ul> <p>If you select (More...), this starts the Type Selection wizard. From this wizard you can select any of the available types.</p> <p>If you select (New Simple Type), this starts the New Simple Type wizard which allows you to create an Anonymous simple type which will be based on an existing type. This can be created locally or globally.</p> <p>If you select (New Complex Type), this starts the New Complex Type wizard which allows you to create an Anonymous complex type which can be derived from an existing base type. This can be created locally or globally.</p> <p>For further information about these types, and examples of their use see the XML Schema Part 0: Primer which can be found on the World Wide Web Consortium (W3C) Web site.</p>
Namespace	Enumerated type	<p>Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references.</p> <p>If &lt;no target namespace&gt; is displayed, a namespace has not been set for this object.</p> <p>If the property is inactive, the message set has not been configured to support namespaces.</p> <p>Where the property is active, namespaces that are available for selection will be displayed in the drop down list.</p>

## Value

Property	Type	Meaning
Default	Button and String	<p>This property provides the default value for the element or attribute.</p> <p>When writing a fixed-length portion of a message (CWF or fixed-length TDS) if the attribute or element is missing from the message tree, the default value will be inserted into the bit stream in order to preserve the message structure.</p> <p>This property is not used by the message broker when parsing.</p> <p>Note that this behavior differs from the XML Schema specification.</p>

Property	Type	Meaning
Fixed	Button and String	<p>If a <i>Fixed</i> value is set for an attribute and a message is received, if the attribute exists in the message, the data in the attribute property of the message must match that held in the fixed value of the attribute definition.</p> <p>If the attribute does not exist in the message, the broker will create the attribute and populate it with the data held in the fixed value of the attribute definition.</p>
Nilable	Check box	Select this if you want the element to be able to be defined as null. This is distinct from being empty where there is no data in the element.
Interpret Value As	Enumerated Type	<p>Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur.</p> <p>The possible values for this property are:</p> <ul style="list-style-type: none"> <li>• <b>None</b> This is the default value and will indicate that the element or attribute does not have any key value associated with it.</li> <li>• <b>Message Set Identity.</b> Specifies that the value of the element or attribute will correspond to the identifier, name or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendants of the enclosing message are defined. This value will remain in force unless a new element or attribute Message Set Identity field is encountered which resets the Message Set Identity value to this new one.</li> <li>• <b>Message Identity.</b> Specifies that the value of the element or attribute will correspond to the name or alias (in that priority order) that is associated with a message and will act as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity will apply until a new element or attribute Message Identity field is encountered at the same level in the tree. The embedded message may be defined in either the current message set or in a message set identified using a Message Set Identity.</li> </ul> <p><b>Note:</b> This property is only applicable when the type of the object is derived from <code>xsd:string</code>.</p>

## Substitution settings

Substitution Groups provide a means by which one element may be substituted for another in a message. The element which can be substituted is called the 'head' element, and the substitution group is the list of elements that may be used in its place. An element can be in at most one substitution group.

Property	Type	Meaning
Final	Enumerated type	<p>You use this property to limit the set of elements which may belong to its substitution group.</p> <ul style="list-style-type: none"> <li>• <b>Empty</b></li> <li>• <b>restriction.</b> Prohibit element substitution by elements whose types are restrictions of the head element's type.</li> <li>• <b>extension.</b> Prohibit element substitution by elements whose types are extensions of the head element's type.</li> <li>• <b>all.</b> Prohibit substitution by any method.</li> </ul>

Property	Type	Meaning
Block	Enumerated type	<p>You use this property to limit the set of elements which may be substituted for this element in a message.</p> <p>Select from:</p> <ul style="list-style-type: none"> <li>• Empty</li> <li>• restriction. Prohibit element substitution by elements whose types are restrictions of the head element's type</li> <li>• extension. Prohibit element substitution by elements whose types are extensions of the head element's type</li> <li>• substitution. Prohibit element substitution by members of the element's substitution group.</li> <li>• #all. Prohibit substitution by any method.</li> </ul>
Substitution Group	Enumerated type	Use this property to specify the name of a 'head' element. Setting this property indicates that this element is a member of the substitution group for the 'head' element.
Abstract	Check box	Select this if you do not want the element to appear in the message, but require one of the members of its substitution group to appear in its place.

## Global group logical properties

Valid children in a global group that depend on both **Composition** and **Content Validation** are shown in "Content Validation properties for complex types" on page 182.

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"> <li>• - the hyphen</li> <li>• _ the underscore</li> <li>• . the period</li> </ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <code>xml</code> or any variant (for example <code>XML</code>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>

Property	Type	Meaning
Composition	Enumerated type	<p>The following only applies to the element content of a complex type and has no effect on the attribute content of a complex type. Select from:</p> <ul style="list-style-type: none"> <li>• Empty</li> <li>• sequence. If you select this option, you can define children that are elements or groups. These children, if present, must appear in the specified order. They can repeat and can be duplicated.</li> <li>• choice. If you select this option, you can define children that are elements or groups. Only one of the defined children of the complex type can be present, but repeating children are allowed. Use this option if you want to model C unions and COBOL REDEFINES in a Custom Wire Format, or an XML DTD element that uses choice in an XML Wire Format, or a SWIFT field that has more than one option.</li> <li>• all. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can only contain elements - groups are not allowed. An all group can only be used at the top level of a complex type - it cannot be a member of another group within a type.</li> <li>• unorderedSet. If you select this option, you can define only elements as children. The elements can repeat but cannot be duplicated. Child elements can appear in any order.</li> <li>• orderedSet. If you select this option, you can define only elements as children. These elements, if present, must appear in the specified order, and they can repeat but cannot be duplicated. This is the default value for new complex types.</li> <li>• message. If you select this option, you can define only messages as children. They can repeat, but they cannot be duplicated. Like choice, only one of the defined children can be present. If the complex type includes more than one message, the bit stream contains the exact length of the embedded message, and is not padded to the length of the longest. Use this option to model multipart messages, which are used in some industry standards, for example, SWIFT. For more information, see the section on multipart messages in “Multipart messages” on page 28.</li> </ul>
Content Validation	Enumerated type	<p><i>Content Validation</i> controls how the broker responds to undeclared content and specifies where the objects that are included within the complex type are defined, if at all. It is used in combination with the <i>Composition</i> property.</p> <p>Options:</p> <ul style="list-style-type: none"> <li>• Closed. The complex type can only contain the child elements that you have added to it.</li> <li>• Open Defined. The complex type can contain any valid element defined within the message set.</li> <li>• Open. The complex type can contain any valid element, not just those that you have added to this complex type.</li> </ul> <p>See “Combinations of Composition and Content Validation” on page 280 for further details of these options.</p>

### Group reference logical properties

Property	Type	Meaning
Reference Name	Enumerated type	The <i>Reference Name</i> is the name of the object that this object is referring to. The objects available to reference can be selected from the drop down list.

## Occurrence properties

Property	Type	Meaning
Min Occurs	Integer	<p>Specify the minimum number of times that the object can repeat. The default is 1.</p> <p>If the value is set to 0, then the object is optional.</p> <p>With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i>, it must be less than or equal to the value in <i>Max Occurs</i>.</p>
Max Occurs	Integer	<p>Specify the maximum number of times that the object can repeat. The default is 1.</p> <p>If this property is not set, then the object can not occur more than once.</p> <p>If this property is set to 0, it will be interpreted as if the object does not exist in the message.</p> <p>It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.</p>

## Key logical properties

There are no properties to show.

## Keyref logical properties

There are no properties to show.

## Local attribute logical properties

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"><li>• - the hyphen</li><li>• _ the underscore</li><li>• . the period</li></ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <i>xml</i> or any variant (for example <i>Xml</i>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>



Property	Type	Meaning
Type	Enumerated type	<p>The Type property constrains the type of data that can be present in the object.</p> <p>There are a limited number of types available directly from the drop down selector. These are;</p> <ul style="list-style-type: none"> <li>• int</li> <li>• string</li> <li>• boolean</li> <li>• hexBinary</li> <li>• dateTime</li> <li>• date</li> <li>• time</li> <li>• decimal</li> <li>• float</li> <li>• (More...)</li> <li>• (New Simple Type)</li> <li>• (New Complex Type)</li> </ul> <p>If you select (More...), this starts the Type Selection wizard. From this wizard you can select any of the available types.</p> <p>If you select (New Simple Type), this starts the New Simple Type wizard which allows you to create an Anonymous simple type which will be based on an existing type. This can be created locally or globally.</p> <p>If you select (New Complex Type), this starts the New Complex Type wizard which allows you to create an Anonymous complex type which can be derived from an existing base type. This can be created locally or globally.</p> <p>For further information about these types, and examples of their use see the XML Schema Part 0: Primer which can be found on the World Wide Web Consortium (W3C) Web site.</p>
Namespace	Enumerated type	<p>Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references.</p> <p>If &lt;no target namespace&gt; is displayed, a namespace has not been set for this object.</p> <p>If the property is inactive, the message set has not been configured to support namespaces.</p> <p>Where the property is active, namespaces that are available for selection will be displayed in the drop down list.</p>

## Value

The *Value* properties are used in conjunction with the *Usage* property in an Attribute Reference or a Local Attribute.

Property	Type	Meaning
Default	Button and String	<p>This property provides the default value for the element or attribute.</p> <p>When writing a fixed-length portion of a message (CWF or fixed-length TDS) if the attribute or element is missing from the message tree, the default value will be inserted into the bit stream in order to preserve the message structure.</p> <p>This property is not used by the message broker when parsing.</p> <p>Note that this behavior differs from the XML Schema specification.</p>
Fixed	Button and String	<p>If a <i>Fixed</i> value is set for an attribute and a message is received, if the attribute exists in the message, the data in the attribute property of the message must match that held in the fixed value of the attribute definition.</p> <p>If the attribute does not exist in the message, the broker will create the attribute and populate it with the data held in the fixed value of the attribute definition.</p>
Interpret Value As	Enumerated Type	<p>Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur.</p> <p>The possible values for this property are:</p> <ul style="list-style-type: none"> <li>• None This is the default value and will indicate that the element or attribute does not have any key value associated with it.</li> <li>• Message Set Identity. Specifies that the value of the element or attribute will correspond to the identifier, name or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value will remain in force unless a new element or attribute Message Set Identity field is encountered which resets the Message Set Identity value to this new one.</li> <li>• Message Identity. Specifies that the value of the element or attribute will correspond to the name or alias (in that priority order) that is associated with a message and will act as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity will apply until a new element or attribute Message Identity field is encountered at the same level in the tree. The embedded message may be defined in either the current message set or in a message set identified using a Message Set Identity.</li> </ul> <p><b>Note:</b> This property is only applicable when the type of the object is derived from <code>xsd:string</code>.</p>

## Usage properties

Property	Type	Meaning
Usage	Enumerated type	<p>The usage property is used in conjunction with the <i>Value</i> property found in an attribute object. The default for the <i>Usage</i> property is optional.</p> <p>Select from;</p> <ul style="list-style-type: none"> <li>• optional. <ul style="list-style-type: none"> <li>– Where the <i>Value</i> property is set to default and no data has been entered in the <i>Value</i> property, the attribute can appear once and can have any value.</li> <li>– Where the <i>Value</i> property is set to default, the attribute can appear once. If it does not appear, its value will be the data that has been entered in the <i>Value</i> property. If it does appear it will be the value given.</li> <li>– Where the <i>Value</i> property is set to fixed, the attribute can appear once. If it does appear, its value must match the data that has been entered in the <i>Value</i> property. If it does not appear its value will be the data that has been entered in the <i>Value</i> property.</li> </ul> </li> <li>• prohibited. The attribute must not appear.</li> <li>• required. <ul style="list-style-type: none"> <li>– Where the <i>Value</i> property is set to default and no data has been entered in the <i>Value</i> property, the attribute must appear once and can have any value.</li> <li>– Where the <i>Value</i> property is set to fixed, the attribute must appear once and it must match the data that has been entered in the <i>Value</i> property.</li> </ul> </li> </ul>

## Local element logical properties

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"> <li>• - the hyphen</li> <li>• _ the underscore</li> <li>• . the period</li> </ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <i>xml</i> or any variant (for example <i>Xml</i>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>

Property	Type	Meaning
Type	Enumerated type	<p>The Type property constrains the type of data that can be present in the object.</p> <p>There are a limited number of types available directly from the drop down selector. These are;</p> <ul style="list-style-type: none"> <li>• int</li> <li>• string</li> <li>• boolean</li> <li>• hexBinary</li> <li>• dateTime</li> <li>• date</li> <li>• time</li> <li>• decimal</li> <li>• float</li> <li>• (More...)</li> <li>• (New Simple Type)</li> <li>• (New Complex Type)</li> </ul> <p>If you select (More...), this starts the Type Selection wizard. From this wizard you can select any of the available types.</p> <p>If you select (New Simple Type), this starts the New Simple Type wizard which allows you to create an Anonymous simple type which will be based on an existing type. This can be created locally or globally.</p> <p>If you select (New Complex Type), this starts the New Complex Type wizard which allows you to create an Anonymous complex type which can be derived from an existing base type. This can be created locally or globally.</p> <p>For further information about these types, and examples of their use see the XML Schema Part 0: Primer which can be found on the World Wide Web Consortium (W3C) Web site.</p>
Namespace	Enumerated type	<p>Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references.</p> <p>If &lt;no target namespace&gt; is displayed, a namespace has not been set for this object.</p> <p>If the property is inactive, the message set has not been configured to support namespaces.</p> <p>Where the property is active, namespaces that are available for selection will be displayed in the drop down list.</p>

## Occurrences

Property	Type	Meaning
Min Occurs	Integer	<p>Specify the minimum number of times that the object can repeat. The default is 1.</p> <p>If the value is set to 0, then the object is optional.</p> <p>With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i>, it must be less than or equal to the value in <i>Max Occurs</i>.</p>

Property	Type	Meaning
Max Occurs	Integer	<p>Specify the maximum number of times that the object can repeat. The default is 1.</p> <p>If this property is not set, then the object can not occur more than once.</p> <p>If this property is set to 0, it will be interpreted as if the object does not exist in the message.</p> <p>It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.</p>

## Value

Property	Type	Meaning
Default	Button and String	<p>This property provides the default value for the element or attribute.</p> <p>When writing a fixed-length portion of a message (CWF or fixed-length TDS) if the attribute or element is missing from the message tree, the default value will be inserted into the bit stream in order to preserve the message structure.</p> <p>This property is not used by the message broker when parsing.</p> <p>Note that this behavior differs from the XML Schema specification.</p>
Fixed	Button and String	<p>If a <i>Fixed</i> value is set for an attribute and a message is received, if the attribute exists in the message, the data in the attribute property of the message must match that held in the fixed value of the attribute definition.</p> <p>If the attribute does not exist in the message, the broker will create the attribute and populate it with the data held in the fixed value of the attribute definition.</p>
Nilable	Check box	<p>Select this if you want the element to be able to be defined as null. This is distinct from being empty where there is no data in the element.</p>
Interpret Value As	Enumerated Type	<p>Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur.</p> <p>The possible values for this property are:</p> <ul style="list-style-type: none"> <li>• None This is the default value and will indicate that the element or attribute does not have any key value associated with it.</li> <li>• Message Set Identity. Specifies that the value of the element or attribute will correspond to the identifier, name or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value will remain in force unless a new element or attribute Message Set Identity field is encountered which resets the Message Set Identity value to this new one.</li> <li>• Message Identity. Specifies that the value of the element or attribute will correspond to the name or alias (in that priority order) that is associated with a message and will act as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity will apply until a new element or attribute Message Identity field is encountered at the same level in the tree. The embedded message may be defined in either the current message set or in a message set identified using a Message Set Identity.</li> </ul> <p><b>Note:</b> This property is only applicable when the type of the object is derived from xsd:string.</p>

## Substitution settings

Substitution Groups provide a means by which one element may be substituted for another in a message. The element which can be substituted is called the 'head' element, and the substitution group is the list of elements that may be used in its place. An element can be in at most one substitution group.

Property	Type	Meaning
Final	Enumerated type	You use this property to limit the set of elements which may belong to its substitution group. <ul style="list-style-type: none"><li>• Empty</li><li>• <i>restriction</i>. Prohibit element substitution by elements whose types are restrictions of the head element's type.</li><li>• <i>extension</i>. Prohibit element substitution by elements whose types are extensions of the head element's type.</li><li>• <i>all</i>. Prohibit substitution by any method.</li></ul>
Block	Enumerated type	You use this property to limit the set of elements which may be substituted for this element in a message.  Select from: <ul style="list-style-type: none"><li>• Empty</li><li>• <i>restriction</i>. Prohibit element substitution by elements whose types are restrictions of the head element's type</li><li>• <i>extension</i>. Prohibit element substitution by elements whose types are extensions of the head element's type</li><li>• <i>substitution</i>. Prohibit element substitution by members of the element's substitution group.</li><li>• <i>#all</i>. Prohibit substitution by any method.</li></ul>
Substitution Group	Enumerated type	Use this property to specify the name of a 'head' element. Setting this property indicates that this element is a member of the substitution group for the 'head' element.
Abstract	Check box	Select this if you do not want the element to appear in the message, but require one of the members of its substitution group to appear in its place.

## Local group logical properties

Valid children in a local group that depend on both **Composition** and **Content Validation** are shown in "Content Validation properties for complex types" on page 182.

Property	Type	Meaning
Composition	Enumerated type	<p>The following only applies to the element content of a complex type and has no effect on the attribute content of a complex type. Select from:</p> <ul style="list-style-type: none"> <li>• Empty</li> <li>• sequence. If you select this option, you can define children that are elements or groups. These children, if present, must appear in the specified order. They can repeat and can be duplicated.</li> <li>• choice. If you select this option, you can define children that are elements or groups. Only one of the defined children of the complex type can be present, but repeating children are allowed. Use this option if you want to model C unions and COBOL REDEFINES in a Custom Wire Format, or an XML DTD element that uses choice in an XML Wire Format, or a SWIFT field that has more than one option.</li> <li>• all. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can only contain elements - groups are not allowed. An all group can only be used at the top level of a complex type - it cannot be a member of another group within a type.</li> <li>• unorderedSet. If you select this option, you can define only elements as children. The elements can repeat but cannot be duplicated. Child elements can appear in any order.</li> <li>• orderedSet. If you select this option, you can define only elements as children. These elements, if present, must appear in the specified order, and they can repeat but cannot be duplicated. This is the default value for new complex types.</li> <li>• message. If you select this option, you can define only messages as children. They can repeat, but they cannot be duplicated. Like choice, only one of the defined children can be present. If the complex type includes more than one message, the bit stream contains the exact length of the embedded message, and is not padded to the length of the longest. Use this option to model multipart messages, which are used in some industry standards, for example, SWIFT. For more information, see the section on multipart messages in “Multipart messages” on page 28.</li> </ul>
Content Validation	Enumerated type	<p><i>Content Validation</i> controls how the broker responds to undeclared content and specifies where the objects that are included within the complex type are defined, if at all. It is used in combination with the <i>Composition</i> property.</p> <p>Options:</p> <ul style="list-style-type: none"> <li>• Closed. The complex type can only contain the child elements that you have added to it.</li> <li>• Open Defined. The complex type can contain any valid element defined within the message set.</li> <li>• Open. The complex type can contain any valid element, not just those that you have added to this complex type.</li> </ul> <p>See “Combinations of Composition and Content Validation” on page 280 for further details of these options.</p>

## Occurrences

Property	Type	Meaning
Min Occurs	Integer	<p>Specify the minimum number of times that the object can repeat. The default is 1.</p> <p>If the value is set to 0, then the object is optional.</p> <p>With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i>, it must be less than or equal to the value in <i>Max Occurs</i>.</p>
Max Occurs	Integer	<p>Specify the maximum number of times that the object can repeat. The default is 1.</p> <p>If this property is not set, then the object can not occur more than once.</p> <p>If this property is set to 0, it will be interpreted as if the object does not exist in the message.</p> <p>It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.</p>

## Message logical properties

This section describes the logical properties of a message.

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"> <li>• - the hyphen</li> <li>• _ the underscore</li> <li>• . the period</li> </ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <i>xml</i> or any variant (for example <i>XML</i>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>
Message Alias	String	<p>Specify an alternative unique value that identifies the message. This property is only required if you are using the Message Identity technique to identify embedded messages, and the bit stream does not contain the actual message name.</p>



## Simple type logical properties

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"> <li>• - the hyphen</li> <li>• _ the underscore</li> <li>• . the period</li> </ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <i>xml</i> or any variant (for example <i>Xml</i>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>
Base Type	Enumerated type	<p>This property only applies to a simple type restriction.</p> <p>You can use this property to select a base type that is used as the starting point to define a new simple type that is derived by setting additional value constraints.</p>
Item Type	Enumerated type	<p>This property only applies to a simple type list.</p> <p>You can use this property to select the type that is used as the item type of the list.</p>
Variety	Enumerated type	<p>This property displays the variety of the simple type you have selected, either atomic, list, or union.</p>

A simple type can also have “Simple type logical value constraints.”

### Simple type logical value constraints:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
---	----------------------------	--	---

<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration</li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>
--	---	--	--

### Unique logical properties

There are no properties to show.

### Wildcard attribute logical properties

Property	Type	Meaning
Namespace	String	<p>Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references.</p> <p>This field is initially blank.</p>

Property	Type	Meaning
Process Content	Enumerated type	<p>If a message contains an attribute that corresponds to a wildcard in the message model, <i>Process Content</i> defines how the attribute is validated.</p> <p>Select from;</p> <ul style="list-style-type: none"> <li>• <i>strict</i>. The parser can only match against attributes declared in the specified namespace.</li> <li>• <i>lax</i>. The parser attempts to match against attributes declared in any accessible namespace. If the specified namespace cannot be found, an error will not be generated.</li> <li>• <i>skip</i>. If you select skip the parser does not perform any validation on the attribute.</li> </ul>

### Wildcard element logical properties

Property	Type	Meaning
Namespace	String	<p>Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references.</p> <p>This field is initially blank.</p>

Property	Type	Meaning
Process Content	Enumerated type	<p>If a message contains an attribute that corresponds to a wildcard in the message model, <i>Process Content</i> defines how the attribute is validated.</p> <p>Select from;</p> <ul style="list-style-type: none"> <li>• <i>strict</i>. The parser can only match against attributes declared in the specified namespace.</li> <li>• <i>lax</i>. The parser attempts to match against attributes declared in any accessible namespace. If the specified namespace cannot be found, an error will not be generated.</li> <li>• <i>skip</i>. If you select <i>skip</i> the parser does not perform any validation on the attribute.</li> </ul>

## Occurrences

Property	Type	Meaning
Min Occurs	Integer	<p>Specify the minimum number of times that the object can repeat. The default is 1.</p> <p>If the value is set to 0, then the object is optional.</p> <p>With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i>, it must be less than or equal to the value in <i>Max Occurs</i>.</p> <p>This property is ignored by brokers that are prior to WebSphere Message Broker Version 6.0.</p>
Max Occurs	Integer	<p>Specify the maximum number of times that the object can repeat. The default is 1.</p> <p>If this property is not set, then the object can not occur more than once.</p> <p>If this property is set to 0, it will be interpreted as if the object does not exist in the message.</p> <p>It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.</p> <p>This property is ignored by brokers that are prior to WebSphere Message Broker Version 6.0.</p>

## Physical properties for message model objects

Property information is available for objects within:

- “Custom Wire Format physical properties for message model objects”
- “XML wire format physical properties for message model objects” on page 210
- “Tagged/delimited string format physical properties for message model objects” on page 218

### Custom Wire Format physical properties for message model objects

Custom wire format physical property information is available for the following objects:

- “Attribute group reference CWF properties” on page 204
- “Attribute reference CWF properties” on page 204

- “Complex type CWF properties” on page 205
- “Element reference CWF properties” on page 205
- “Global attribute CWF properties” on page 206
- “Global attribute group CWF properties” on page 206
- “Global element CWF properties” on page 206
- “Global group CWF properties” on page 206
- “Group reference CWF properties” on page 206
- “Key CWF properties” on page 207
- “Keyref CWF properties” on page 207
- “Local element CWF properties” on page 208
- “Local group CWF properties” on page 209
- “Message CWF properties” on page 209
- “Simple type CWF properties” on page 209
- “Unique CWF properties” on page 209
- “Wildcard attribute CWF properties” on page 210
- “Wildcard element CWF properties” on page 210

**Attribute group reference CWF properties:**

There are no properties to show.

**Attribute reference CWF properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types	Boolean types	DateTime types	Decimal types
<ul style="list-style-type: none"> <li>- base64Binary</li> <li>- hexBinary</li> </ul>	<ul style="list-style-type: none"> <li>- boolean</li> </ul>	<ul style="list-style-type: none"> <li>- date</li> <li>- dateTime</li> <li>- gDay</li> <li>- gMonth</li> <li>- gMonthDay</li> <li>- gYear</li> <li>- gYearMonth</li> <li>- time</li> </ul>	<ul style="list-style-type: none"> <li>- decimal</li> <li>- integer</li> <li>- negativeInteger</li> <li>- nonNegativeInteger</li> <li>- nonPositiveInteger</li> <li>- positiveInteger</li> </ul>

<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>
<p><b>Note:</b></p> <p>1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.</p>			

#### Complex type CWF properties:

There are no properties to show.

#### Element reference CWF properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

<p>Binary types</p> <ul style="list-style-type: none"> <li>- base64Binary</li> <li>- hexBinary</li> </ul>	<p>Boolean types</p> <ul style="list-style-type: none"> <li>- boolean</li> </ul>	<p>DateTime types</p> <ul style="list-style-type: none"> <li>- date</li> <li>- dateTime</li> <li>- gDay</li> <li>- gMonth</li> <li>- gMonthDay</li> <li>- gYear</li> <li>- gYearMonth</li> <li>- time</li> </ul>	<p>Decimal types</p> <ul style="list-style-type: none"> <li>- decimal</li> <li>- integer</li> <li>- negativeInteger</li> <li>- nonNegativeInteger</li> <li>- nonPositiveInteger</li> <li>- positiveInteger</li> </ul>
---	--	--	---

<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>
--	---	--	--

**Note:**

1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.

**Global attribute CWF properties:**

There are no properties to show.

**Global attribute group CWF properties:**

There are no properties to show.

**Global element CWF properties:**

There are no properties to show.

**Global group CWF properties:**

There are no properties to show.

**Group reference CWF properties:**

**Byte alignment**

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>

Property	Type	Meaning
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.  Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.  If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.

### Key CWF properties:

There are no properties to show.

### Keyref CWF properties:

There are no properties to show.

### Local attribute CWF properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
---	----------------------------	--	---

<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>
--	---	--	--

**Note:**

1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.

**Local element CWF properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

<p>Binary types</p> <ul style="list-style-type: none"> <li>- base64Binary</li> <li>- hexBinary</li> </ul>	<p>Boolean types</p> <ul style="list-style-type: none"> <li>- boolean</li> </ul>	<p>DateTime types</p> <ul style="list-style-type: none"> <li>- date</li> <li>- dateTime</li> <li>- gDay</li> <li>- gMonth</li> <li>- gMonthDay</li> <li>- gYear</li> <li>- gYearMonth</li> <li>- time</li> </ul>	<p>Decimal types</p> <ul style="list-style-type: none"> <li>- decimal</li> <li>- integer</li> <li>- negativeInteger</li> <li>- nonNegativeInteger</li> <li>- nonPositiveInteger</li> <li>- positiveInteger</li> </ul>
<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>



**Note:**

1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.

**Local group CWF properties:  
Byte alignment**

Property	Type	Meaning
Byte Alignment	Enumerated Type	Specify how the object is aligned from the start of the message. Select one of: <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

**Repeat**

Property	Type	Meaning
Repeat Reference	Enumerated Type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.  Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.  If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.

**Message CWF properties:**

There are no properties to show.

**Simple type CWF properties:**

There are no properties to show.

**Unique CWF properties:**

There are no properties to show.

**Wildcard attribute CWF properties:**

There are no properties to show.

**Wildcard element CWF properties:**

There are no properties to show.

**XML wire format physical properties for message model objects**

XML wire format physical property information is available for the following objects:

- “Attribute group reference XML properties”
- “Attribute reference XML properties”
- “Complex type XML properties” on page 211
- “Element reference XML properties” on page 211
- “Global attribute XML properties” on page 212
- “Global attribute group XML properties” on page 213
- “Global element XML properties” on page 213
- “Global group XML properties” on page 213
- “Group reference XML properties” on page 213
- “Key XML properties” on page 213
- “Keyref XML properties” on page 214
- “Local attribute XML properties” on page 214
- “Local element XML properties” on page 214
- “Local group XML properties” on page 215
- “Message XML properties” on page 215
- “Simple type XML properties” on page 218
- “Unique XML properties” on page 218
- “Wildcard attribute XML properties” on page 218
- “Wildcard element XML properties” on page 218

**Attribute group reference XML properties:**

There are no properties to show.

**Attribute reference XML properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
Float types - double - float	Integer types - byte - int - long - short - unsignedByte - unsignedInt - unsignedLong - unsignedShort	Interval types - duration <sup>1</sup>	String types - anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token
<b>Note:</b> 1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.			

**Complex type XML properties:**

There are no properties to show.

**Element reference XML properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
---	----------------------------	--	---

<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>
--	---	--	--

**Note:**

1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.

**Global attribute XML properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

<p>Binary types</p> <ul style="list-style-type: none"> <li>- base64Binary</li> <li>- hexBinary</li> </ul>	<p>Boolean types</p> <ul style="list-style-type: none"> <li>- boolean</li> </ul>	<p>DateTime types</p> <ul style="list-style-type: none"> <li>- date</li> <li>- dateTime</li> <li>- gDay</li> <li>- gMonth</li> <li>- gMonthDay</li> <li>- gYear</li> <li>- gYearMonth</li> <li>- time</li> </ul>	<p>Decimal types</p> <ul style="list-style-type: none"> <li>- decimal</li> <li>- integer</li> <li>- negativeInteger</li> <li>- nonNegativeInteger</li> <li>- nonPositiveInteger</li> <li>- positiveInteger</li> </ul>
<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>

**Note:**

1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.

**Global attribute group XML properties:**

There are no properties to show.

**Global element XML properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
Float types - double - float	Integer types - byte - int - long - short - unsignedByte - unsignedInt - unsignedLong - unsignedShort	Interval types - duration <sup>1</sup>	String types - anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

**Note:**

1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.

**Global group XML properties:**

There are no properties to show.

**Group reference XML properties:**

There are no properties to show.

**Key XML properties:**

There are no properties to show.

**Keyref XML properties:**

There are no properties to show.

**Local attribute XML properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

<p>Binary types</p> <ul style="list-style-type: none"> <li>- base64Binary</li> <li>- hexBinary</li> </ul>	<p>Boolean types</p> <ul style="list-style-type: none"> <li>- boolean</li> </ul>	<p>DateTime types</p> <ul style="list-style-type: none"> <li>- date</li> <li>- dateTime</li> <li>- gDay</li> <li>- gMonth</li> <li>- gMonthDay</li> <li>- gYear</li> <li>- gYearMonth</li> <li>- time</li> </ul>	<p>Decimal types</p> <ul style="list-style-type: none"> <li>- decimal</li> <li>- integer</li> <li>- negativeInteger</li> <li>- nonNegativeInteger</li> <li>- nonPositiveInteger</li> <li>- positiveInteger</li> </ul>
<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>

**Note:**

1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.

**Local element XML properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
Float types - double - float	Integer types - byte - int - long - short - unsignedByte - unsignedInt - unsignedLong - unsignedShort	Interval types - duration <sup>1</sup>	String types - anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token
<b>Note:</b> 1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.			

#### Local group XML properties:

There are no properties to show.

#### Message XML properties: Namespace schema locations

This property is only active if namespaces have been enabled.

Property	Type	Meaning
Namespace URI	String	A unique string, usually in the form of a URL that identifies the schema for this  If namespaces have not been enabled, this property will display <no target namespace>.
		This property will override the same property at the message set level.
Schema location	String	Enter the location of the schema for the associated namespace name that will be used to validate objects within the namespace.

## XML declarations

Property	Type	Meaning
Output Namespace Declaration	Enumerated Type	<p>The <i>Output Namespace Declaration</i> property controls where the namespace declarations will be placed in the output XML document.</p> <p>Select from:</p> <ul style="list-style-type: none"> <li>• At start of document. Declarations for all of the entries in the <i>Namespace schema locations</i> table above will be output as attributes of the message in the output XML document. The disadvantage of this option is that in some cases unnecessary declarations may be output.</li> <li>• As required. Declarations will only be output when required by an element or attribute that is in that namespace. The disadvantage of this option is that the same namespace declaration may need to be output more than once in the output XML document.</li> </ul> <p>The default option is At start of document.</p> <p>This property is only active if namespaces are enabled for this message set.</p>

## XML document type settings

Property	Type	Meaning
DOCTYPE System ID	String	<p>Specify the System ID for DOCTYPE external DTD subset. It overrides the equivalent message set property setting for this particular message.</p> <p>If the message set property <i>Suppress DOCTYPE</i> is set to Yes, this parameter is ignored and cannot be changed (the field is disabled) .</p> <p>The default value is the value that you specified for the <i>DOCTYPE System ID</i> property for the message set.</p>
DOCTYPE Public ID	String	<p>Specify the Public ID for DOCTYPE external DTD subset. It overrides the equivalent message set property setting for this particular message.</p> <p>If the message set property <i>Suppress DOCTYPE</i> is set to Yes, this parameter is ignored and cannot be changed (the field is disabled) .</p> <p>The default value is the value that you specified for the <i>DOCTYPE Public ID</i> property for the message set.</p>
DOCTYPE Text	String	<p>Enter optional additional text to include within the DOCTYPE. It overrides the message set property for this particular message.</p> <p>If the message set property <i>Suppress DOCTYPE</i> is set to Yes, this parameter is ignored and cannot be changed (the field is disabled) .</p> <p>For more information, see “In-line DTDs and the DOCTYPE text property” on page 162.</p> <p>The default value is the value that you specified for the <i>DOCTYPE Text</i> property for the message set.</p>

Property	Type	Meaning
Root Tag Name	String	<p>Specify the name of the root tag for a message bit stream XML document. It overrides the message set property set for this message.</p> <p>The default value is the value that you specified for the <i>Root Tag Name</i> property for the message set.</p>



## Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li> <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element.            If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.         </li> <li> <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>.            If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name</i>, <i>ID Attribute Name</i>, <i>ID Attribute Value</i> to the same values:           <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element’s identifier. If the element’s identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is <i>id</i>.</p>
Namespace	String	<p>Enter the namespace associated with the <i>ID Attribute</i>.</p>
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is the identifier of the child.</p>

**Simple type XML properties:**

There are no properties to show.

**Unique XML properties:**

There are no properties to show.

**Wildcard attribute XML properties:**

There are no properties to show.

**Wildcard element XML properties:**

There are no properties to show.

**Tagged/delimited string format physical properties for message model objects**

TDS format physical property information is available for the following objects:

- “Attribute group reference TDS properties”
- “Attribute reference TDS properties”
- “Complex type TDS properties” on page 219
- “Element reference TDS properties” on page 222
- “Global attribute TDS properties” on page 223
- “Global attribute group TDS properties” on page 223
- “Global element TDS properties” on page 223
- “Global group TDS properties” on page 224
- “Group reference TDS properties” on page 227
- “Key TDS properties” on page 227
- “Keyref TDS properties” on page 228
- “Local attribute TDS properties” on page 228
- “Local element TDS properties” on page 228
- “Local group TDS properties” on page 229
- “Message TDS properties” on page 232
- “Simple type TDS properties” on page 232
- “Unique TDS properties” on page 233
- “Wildcard attribute TDS properties” on page 233
- “Wildcard element TDS properties” on page 233

**Attribute group reference TDS properties:**

There are no properties to show.

**Attribute reference TDS properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
Float types - double - float	Integer types - byte - int - long - short - unsignedByte - unsignedInt - unsignedLong - unsignedShort	Interval types - duration <sup>1</sup>	String types - anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token
<b>Note:</b> 1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.			

### Complex type TDS properties: Field Identification

If the complex type is based on a global group, the TDS properties listed below will actually be located on the global group. If this is the case, any changes to these properties will be applied to the global group, and will affect all references to the group (including any other complex types which are based on it).

Property	Type	Meaning
Data Element Separation	Enumerated Type	<p>Specify the method used to separate the data elements within the type. Select one of the following values:</p> <ul style="list-style-type: none"> <li>• Tagged Delimited. This value indicates that all elements within the complex type are identified by a tag, and separated by the value specified in the optional <i>Delimiter</i> property (if specified). You must set the <i>Tag</i> property for all child elements of simple type, and you may set the <i>Delimiter</i> property to a non-empty value. See “Global element TDS properties” on page 223. You must also set the <i>Tag Data Separator</i> or the <i>Length of Tag</i> properties.</li> <li>• Tagged Fixed Length. This value indicates that each element is identified by a tag, and the data has a fixed length. There are no delimiters. You must set the <i>Tag</i> property for each of the child elements of this complex type, and each child element must have a <i>Length</i> or <i>Length Reference</i> property assigned to it. You must also set the <i>Tag Data Separator</i> or the <i>Length of Tag</i> properties.</li> <li>• Tagged Encoded Length. This value indicates that all elements within the complex type are separated by a tag, and a length field follows each tag. There are no delimiters. The tag can be fixed length, as set by <i>Length of Tag</i>, or variable length delimited by the <i>Tag Data Separator</i>. You must also set <i>Length Of Encoded Length</i> so that the parser knows the size of the length field, and set <i>Extra Chars in Encoded Length</i> to tell the parser how many to subtract from the value in <i>Length Of Encoded Length</i> to get the actual length of data that follows the length field.</li> </ul> <p>This method provides a more flexible way of handling ACORD AL3 standard messages than Fixed Length AL3, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard.</p> <ul style="list-style-type: none"> <li>• All Elements Delimited. This value indicates that all elements within the complex type are separated by a delimiter. You must set the value in the <i>Delimiter</i> property.</li> <li>• Variable Length Elements Delimited. This value indicates that some of the elements within the complex type might be of variable length: if they are, they must be delimited by the value specified in the <i>Delimiter</i> property.</li> <li>• Use Data Pattern. This value indicates that the parser determines the elements by matching the data with the regular expression set in the element or type member <i>Data Pattern</i> property. See “Message definition file properties” on page 174.</li> <li>• Fixed Length. This value indicates that all elements within the complex type are fixed length. The next data element is accessed by adding the value of the <i>Length</i> property to the offset (see “Global element TDS properties” on page 223). If you set the <i>Data Element Separation</i> property of a complex type to Fixed Length, you must also set the <i>Data Element Separation</i> property of all complex children of this type to Fixed Length. Each child element must have a <i>Length</i> or <i>Length Reference</i> property assigned to it.</li> <li>• Fixed Length AL3. This value has a similar meaning to the separation type Fixed Length, but also indicates to the parser that a number of predefined rules with regard to missing optional elements, encoded lengths, and versioning must be applied. If you set the <i>Data Element Separation</i> property of a complex type to Fixed Length AL3, you must also set the <i>Data Element Separation</i> property of all complex children of this type to Fixed Length AL3.</li> <li>• Undefined. This value is set automatically if you set the <i>Type Composition</i> property of a complex type to Message, and you cannot change it to any other value.</li> </ul> <p>Do not set the <i>Type Composition</i> property to Empty, Choice, Unordered Set, Ordered Set, Sequence, or Simple Unordered Set. If you do, you will be unable to check in the type.</p>
Group Indicator	String	Specify the value of a special character or string that precedes the data belonging to a group or complex type within the bit stream.

Property	Type	Meaning
Group Terminator	String	Specify the value of a special character or string that terminates data belonging to a group or a complex type within the bit stream.
Delimiter	String	Specify the value of a special character or string that specifies the delimiter used between data elements.  This property applies only to the delimited <i>Data Element Separation</i> methods (Tagged Delimited, All Elements Delimited, and Variable Elements Delimited).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select if you want delimiters to be suppressed for elements that are missing within a message. Select from: <ul style="list-style-type: none"> <li>• End Of Type. Use this option to suppress the delimiter when an element is missing. For example, if the model has been defined to have up to 3 elements and only 2 are present, the last delimiter can be omitted from the message.</li> <li>• Never. Use this option to ensure that even if optional elements are not present, all delimiters will be written out. This option should be used when the delimiter used to delimit parent and child objects is the same. For example, if an optional child element is missing, message processing applications could not tell where the child elements in a message ended and the next parent element started if the delimiters are all the same.</li> </ul>
Observe Element Length	Check box	Applicable when <i>Data Element Separation</i> is All Elements Delimited and tells the TDS parser to take any <i>Length</i> property of child elements or attributes into account. The default value depends on the setting of the <i>Messaging Standard</i> property (at the message set level) and <i>Data Element Separation</i> properties. <ul style="list-style-type: none"> <li>• Where <i>Data Element Separation</i> is All Elements Delimited and the <i>Messaging Standard</i> is set to TLOG , this property should be set. For all other messaging standards it should not be set.</li> <li>• Where <i>Data Element Separation</i> is Tagged Delimited this property should not be set.</li> <li>• Where <i>Data Element Separation</i> is Tagged Fixed Length, Fixed Length, Fixed Length AL3, or Variable Length Elements Delimited this property will be set and is disabled.</li> <li>• For all other data element separation methods, this property is not set and is disabled.</li> </ul> Any other combination will generate a task list warning.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The <i>Tag Data Separator</i> and <i>Length of Tag</i> properties are mutually exclusive.  If you set the property <i>Tag Data Separator</i> , it overrides <i>Length of Tag</i> .  This property applies only to the tagged <i>Data Element Separation</i> methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).
Length of Tag	Integer	Specify the length of a tag value. When the message is parsed, this allows tags to be extracted from the bit stream if the <i>Tag Data Separator</i> property is not set.  The <i>Tag Data Separator</i> and <i>Length of Tag</i> properties are mutually exclusive. If you set the property <i>Tag Data Separator</i> , it overrides this value.  This property applies only to the tagged <i>Data Element Separation</i> methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).

Property	Type	Meaning
Length of Encoded Length	Integer	<p>Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647.</p> <p>You must set this property if you have set <i>Data Element Separation</i> property to Tagged Encoded Length; it is not valid otherwise.</p> <p>The actual number of data characters parsed also depends on the value of the <i>Extra Chars in Encoded Length</i> property.</p>
Extra Chars in Encoded Length	Integer	<p>(Only valid if the <i>Data Element Separation</i> method is set to Tagged Encoded Length.) Specifies the number of extra characters included in the value found in the length field. (For example, the value in the length might include the size of the length field itself as well as the size of the data field, or it might be the total size of the tag, length, and data fields.)</p> <p>Enter a value from 0 to 2147483647. The parser subtracts this number from the number found in the length field to get the number of data characters that follow the length field.</p> <p>You must set this property if you have set <i>Data Element Separation</i> property to Tagged Encoded Length and the actual number of data characters is less than the value found in the length field.</p>

#### Element reference TDS properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

<p>Binary types</p> <ul style="list-style-type: none"> <li>- base64Binary</li> <li>- hexBinary</li> </ul>	<p>Boolean types</p> <ul style="list-style-type: none"> <li>- boolean</li> </ul>	<p>DateTime types</p> <ul style="list-style-type: none"> <li>- date</li> <li>- dateTime</li> <li>- gDay</li> <li>- gMonth</li> <li>- gMonthDay</li> <li>- gYear</li> <li>- gYearMonth</li> <li>- time</li> </ul>	<p>Decimal types</p> <ul style="list-style-type: none"> <li>- decimal</li> <li>- integer</li> <li>- negativeInteger</li> <li>- nonNegativeInteger</li> <li>- nonPositiveInteger</li> <li>- positiveInteger</li> </ul>
<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>

**Note:**

1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.

**Global attribute TDS properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
Float types - double - float	Integer types - byte - int - long - short - unsignedByte - unsignedInt - unsignedLong - unsignedShort	Interval types - duration <sup>1</sup>	String types - anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

**Note:**

1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.

**Global attribute group TDS properties:**

There are no properties to show.

**Global element TDS properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

<p>Binary types</p> <ul style="list-style-type: none"> <li>- base64Binary</li> <li>- hexBinary</li> </ul>	<p>Boolean types</p> <ul style="list-style-type: none"> <li>- boolean</li> </ul>	<p>DateTime types</p> <ul style="list-style-type: none"> <li>- date</li> <li>- dateTime</li> <li>- gDay</li> <li>- gMonth</li> <li>- gMonthDay</li> <li>- gYear</li> <li>- gYearMonth</li> <li>- time</li> </ul>	<p>Decimal types</p> <ul style="list-style-type: none"> <li>- decimal</li> <li>- integer</li> <li>- negativeInteger</li> <li>- nonNegativeInteger</li> <li>- nonPositiveInteger</li> <li>- positiveInteger</li> </ul>
<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>
<p><b>Note:</b></p> <p>1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.</p>			

**Global group TDS properties:**



## Field Identification

Property	Type	Meaning
Data Element Separation	Enumerated Type	<p>Specify the method used to separate the data elements within the type. Select one of the following values:</p> <ul style="list-style-type: none"> <li>• Tagged Delimited. This value indicates that all elements within the complex type are identified by a tag, and separated by the value specified in the optional <i>Delimiter</i> property (if specified). You must set the <i>Tag</i> property for all child elements of simple type, and you may set the <i>Delimiter</i> property to a non-empty value. See “Global element TDS properties” on page 223. You must also set the <i>Tag Data Separator</i> or the <i>Length of Tag</i> properties.</li> <li>• Tagged Fixed Length. This value indicates that each element is identified by a tag, and the data has a fixed length. There are no delimiters. You must set the <i>Tag</i> property for each of the child elements of this complex type, and each child element must have a <i>Length</i> or <i>Length Reference</i> property assigned to it. You must also set the <i>Tag Data Separator</i> or the <i>Length of Tag</i> properties.</li> <li>• Tagged Encoded Length. This value indicates that all elements within the complex type are separated by a tag, and a length field follows each tag. There are no delimiters. The tag can be fixed length, as set by <i>Length of Tag</i>, or variable length delimited by the <i>Tag Data Separator</i>. You must also set <i>Length Of Encoded Length</i> so that the parser knows the size of the length field, and set <i>Extra Chars in Encoded Length</i> to tell the parser how many to subtract from the value in <i>Length Of Encoded Length</i> to get the actual length of data that follows the length field.  This method provides a more flexible way of handling ACORD AL3 standard messages than Fixed Length AL3, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard.</li> <li>• All Elements Delimited. This value indicates that all elements within the complex type are separated by a delimiter. You must set the value in the <i>Delimiter</i> property.</li> <li>• Variable Length Elements Delimited. This value indicates that some of the elements within the complex type might be of variable length: if they are, they must be delimited by the value specified in the <i>Delimiter</i> property.</li> <li>• Use Data Pattern. This value indicates that the parser determines the elements by matching the data with the regular expression set in the element or type member <i>Data Pattern</i> property. See “Message definition file properties” on page 174.</li> <li>• Fixed Length. This value indicates that all elements within the complex type are fixed length. The next data element is accessed by adding the value of the <i>Length</i> property to the offset (see “Global element TDS properties” on page 223). If you set the <i>Data Element Separation</i> property of a complex type to Fixed Length, you must also set the <i>Data Element Separation</i> property of all complex children of this type to Fixed Length. Each child element must have a <i>Length</i> or <i>Length Reference</i> property assigned to it.</li> <li>• Fixed Length AL3. This value has a similar meaning to the separation type Fixed Length, but also indicates to the parser that a number of predefined rules with regard to missing optional elements, encoded lengths, and versioning must be applied. If you set the <i>Data Element Separation</i> property of a complex type to Fixed Length AL3, you must also set the <i>Data Element Separation</i> property of all complex children of this type to Fixed Length AL3.</li> <li>• Undefined. This value is set automatically if you set the <i>Type Composition</i> property of a complex type to Message, and you cannot change it to any other value.</li> </ul> <p>Do not set the <i>Type Composition</i> property to Empty, Choice, Unordered Set, Ordered Set, Sequence, or Simple Unordered Set. If you do, you will be unable to check in the type.</p>

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character or string that precedes the data belonging to a group or complex type within the bit stream.
Group Terminator	String	Specify the value of a special character or string that terminates data belonging to a group or a complex type within the bit stream.
Delimiter	String	Specify the value of a special character or string that specifies the delimiter used between data elements.  This property applies only to the delimited <i>Data Element Separation</i> methods (Tagged Delimited, All Elements Delimited, and Variable Elements Delimited).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select if you want delimiters to be suppressed for elements that are missing within a message. Select from: <ul style="list-style-type: none"> <li>• End Of Type. Use this option to suppress the delimiter when an element is missing. For example, if the model has been defined to have up to 3 elements and only 2 are present, the last delimiter can be omitted from the message.</li> <li>• Never. Use this option to ensure that even if optional elements are not present, all delimiters will be written out. This option should be used when the delimiter used to delimit parent and child objects is the same. For example, if an optional child element is missing, message processing applications could not tell where the child elements in a message ended and the next parent element started if the delimiters are all the same.</li> </ul>
Observe Element Length	Check box	Applicable when <i>Data Element Separation</i> is All Elements Delimited and tells the TDS parser to take any <i>Length</i> property of child elements or attributes into account. The default value depends on the setting of the <i>Messaging Standard</i> property (at the message set level) and <i>Data Element Separation</i> properties. <ul style="list-style-type: none"> <li>• Where <i>Data Element Separation</i> is All Elements Delimited and the <i>Messaging Standard</i> is set to TLOG , this property should be set. For all other messaging standards it should not be set.</li> <li>• Where <i>Data Element Separation</i> is Tagged Delimited this property should not be set.</li> <li>• Where <i>Data Element Separation</i> is Tagged Fixed Length, Fixed Length, Fixed Length AL3, or Variable Length Elements Delimited this property will be set and is disabled.</li> <li>• For all other data element separation methods, this property is not set and is disabled.</li> </ul> Any other combination will generate a task list warning.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The <i>Tag Data Separator</i> and <i>Length of Tag</i> properties are mutually exclusive.  If you set the property <i>Tag Data Separator</i> , it overrides <i>Length of Tag</i> .  This property applies only to the tagged <i>Data Element Separation</i> methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).
Length of Tag	Integer	Specify the length of a tag value. When the message is parsed, this allows tags to be extracted from the bit stream if the <i>Tag Data Separator</i> property is not set.  The <i>Tag Data Separator</i> and <i>Length of Tag</i> properties are mutually exclusive. If you set the property <i>Tag Data Separator</i> , it overrides this value.  This property applies only to the tagged <i>Data Element Separation</i> methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).

Property	Type	Meaning
Length of Encoded Length	Integer	<p>Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647.</p> <p>You must set this property if you have set <i>Data Element Separation</i> property to Tagged Encoded Length; it is not valid otherwise.</p> <p>The actual number of data characters parsed also depends on the value of the <i>Extra Chars in Encoded Length</i> property.</p>
Extra Chars in Encoded Length	Integer	<p>(Only valid if the <i>Data Element Separation</i> method is set to Tagged Encoded Length.) Specifies the number of extra characters included in the value found in the length field. (For example, the value in the length might include the size of the length field itself as well as the size of the data field, or it might be the total size of the tag, length, and data fields.)</p> <p>Enter a value from 0 to 2147483647. The parser subtracts this number from the number found in the length field to get the number of data characters that follow the length field.</p> <p>You must set this property if you have set <i>Data Element Separation</i> property to Tagged Encoded Length and the actual number of data characters is less than the value found in the length field.</p>

**Group reference TDS properties:  
Field identification**

Property	Type	Meaning
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the <i>Data Element Separation</i> of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.</p>

**Physical representation**

Property	Type	Meaning
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>

**Key TDS properties:**

There are no properties to show.

**Keyref TDS properties:**

There are no properties to show.

**Local attribute TDS properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
Float types - double - float	Integer types - byte - int - long - short - unsignedByte - unsignedInt - unsignedLong - unsignedShort	Interval types - duration <sup>1</sup>	String types - anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token
<b>Note:</b> 1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.			

**Local element TDS properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

<p>Binary types</p> <ul style="list-style-type: none"> <li>- base64Binary</li> <li>- hexBinary</li> </ul>	<p>Boolean types</p> <ul style="list-style-type: none"> <li>- boolean</li> </ul>	<p>DateTime types</p> <ul style="list-style-type: none"> <li>- date</li> <li>- dateTime</li> <li>- gDay</li> <li>- gMonth</li> <li>- gMonthDay</li> <li>- gYear</li> <li>- gYearMonth</li> <li>- time</li> </ul>	<p>Decimal types</p> <ul style="list-style-type: none"> <li>- decimal</li> <li>- integer</li> <li>- negativeInteger</li> <li>- nonNegativeInteger</li> <li>- nonPositiveInteger</li> <li>- positiveInteger</li> </ul>
<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>
<p><b>Note:</b></p> <p>1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.</p>			

**Local group TDS properties:**

## Field Identification

Property	Type	Meaning
Data Element Separation	Enumerated Type	<p>Specify the method used to separate the data elements within the type. Select one of the following values:</p> <ul style="list-style-type: none"> <li>• <b>Tagged Delimited.</b> This value indicates that all elements within the complex type are identified by a tag, and separated by the value specified in the optional <i>Delimiter</i> property (if specified). You must set the <i>Tag</i> property for all child elements of simple type, and you may set the <i>Delimiter</i> property to a non-empty value. See “Global element TDS properties” on page 223. You must also set the <i>Tag Data Separator</i> or the <i>Length of Tag</i> properties.</li> <li>• <b>Tagged Fixed Length.</b> This value indicates that each element is identified by a tag, and the data has a fixed length. There are no delimiters. You must set the <i>Tag</i> property for each of the child elements of this complex type, and each child element must have a <i>Length</i> or <i>Length Reference</i> property assigned to it. You must also set the <i>Tag Data Separator</i> or the <i>Length of Tag</i> properties.</li> <li>• <b>Tagged Encoded Length.</b> This value indicates that all elements within the complex type are separated by a tag, and a length field follows each tag. There are no delimiters. The tag can be fixed length, as set by <i>Length of Tag</i>, or variable length delimited by the <i>Tag Data Separator</i>. You must also set <i>Length Of Encoded Length</i> so that the parser knows the size of the length field, and set <i>Extra Chars in Encoded Length</i> to tell the parser how many to subtract from the value in <i>Length Of Encoded Length</i> to get the actual length of data that follows the length field.  This method provides a more flexible way of handling ACORD AL3 standard messages than Fixed Length AL3, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard.</li> <li>• <b>All Elements Delimited.</b> This value indicates that all elements within the complex type are separated by a delimiter. You must set the value in the <i>Delimiter</i> property.</li> <li>• <b>Variable Length Elements Delimited.</b> This value indicates that some of the elements within the complex type might be of variable length: if they are, they must be delimited by the value specified in the <i>Delimiter</i> property.</li> <li>• <b>Use Data Pattern.</b> This value indicates that the parser determines the elements by matching the data with the regular expression set in the element or type member <i>Data Pattern</i> property. See “Message definition file properties” on page 174.</li> <li>• <b>Fixed Length.</b> This value indicates that all elements within the complex type are fixed length. The next data element is accessed by adding the value of the <i>Length</i> property to the offset (see “Global element TDS properties” on page 223). If you set the <i>Data Element Separation</i> property of a complex type to Fixed Length, you must also set the <i>Data Element Separation</i> property of all complex children of this type to Fixed Length. Each child element must have a <i>Length</i> or <i>Length Reference</i> property assigned to it.</li> <li>• <b>Fixed Length AL3.</b> This value has a similar meaning to the separation type Fixed Length, but also indicates to the parser that a number of predefined rules with regard to missing optional elements, encoded lengths, and versioning must be applied. If you set the <i>Data Element Separation</i> property of a complex type to Fixed Length AL3, you must also set the <i>Data Element Separation</i> property of all complex children of this type to Fixed Length AL3.</li> <li>• <b>Undefined.</b> This value is set automatically if you set the <i>Type Composition</i> property of a complex type to Message, and you cannot change it to any other value.</li> </ul> <p>Do not set the <i>Type Composition</i> property to Empty, Choice, Unordered Set, Ordered Set, Sequence, or Simple Unordered Set. If you do, you will be unable to check in the type.</p>

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character or string that precedes the data belonging to a group or complex type within the bit stream.
Group Terminator	String	Specify the value of a special character or string that terminates data belonging to a group or a complex type within the bit stream.
Delimiter	String	Specify the value of a special character or string that specifies the delimiter used between data elements.  This property applies only to the delimited <i>Data Element Separation</i> methods (Tagged Delimited, All Elements Delimited, and Variable Elements Delimited).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select if you want delimiters to be suppressed for elements that are missing within a message. Select from: <ul style="list-style-type: none"> <li>• End Of Type. Use this option to suppress the delimiter when an element is missing. For example, if the model has been defined to have up to 3 elements and only 2 are present, the last delimiter can be omitted from the message.</li> <li>• Never. Use this option to ensure that even if optional elements are not present, all delimiters will be written out. This option should be used when the delimiter used to delimit parent and child objects is the same. For example, if an optional child element is missing, message processing applications could not tell where the child elements in a message ended and the next parent element started if the delimiters are all the same.</li> </ul>
Observe Element Length	Check box	Applicable when <i>Data Element Separation</i> is All Elements Delimited and tells the TDS parser to take any <i>Length</i> property of child elements or attributes into account. The default value depends on the setting of the <i>Messaging Standard</i> property (at the message set level) and <i>Data Element Separation</i> properties. <ul style="list-style-type: none"> <li>• Where <i>Data Element Separation</i> is All Elements Delimited and the <i>Messaging Standard</i> is set to TLOG , this property should be set. For all other messaging standards it should not be set.</li> <li>• Where <i>Data Element Separation</i> is Tagged Delimited this property should not be set.</li> <li>• Where <i>Data Element Separation</i> is Tagged Fixed Length, Fixed Length, Fixed Length AL3, or Variable Length Elements Delimited this property will be set and is disabled.</li> <li>• For all other data element separation methods, this property is not set and is disabled.</li> </ul> Any other combination will generate a task list warning.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The <i>Tag Data Separator</i> and <i>Length of Tag</i> properties are mutually exclusive.  If you set the property <i>Tag Data Separator</i> , it overrides <i>Length of Tag</i> .  This property applies only to the tagged <i>Data Element Separation</i> methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).
Length of Tag	Integer	Specify the length of a tag value. When the message is parsed, this allows tags to be extracted from the bit stream if the <i>Tag Data Separator</i> property is not set.  The <i>Tag Data Separator</i> and <i>Length of Tag</i> properties are mutually exclusive. If you set the property <i>Tag Data Separator</i> , it overrides this value.  This property applies only to the tagged <i>Data Element Separation</i> methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).

Property	Type	Meaning
Length of Encoded Length	Integer	<p>Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647.</p> <p>You must set this property if you have set <i>Data Element Separation</i> property to Tagged Encoded Length; it is not valid otherwise.</p> <p>The actual number of data characters parsed also depends on the value of the <i>Extra Chars in Encoded Length</i> property.</p>
Extra Chars in Encoded Length	Integer	<p>(Only valid if the <i>Data Element Separation</i> method is set to Tagged Encoded Length.) Specifies the number of extra characters included in the value found in the length field. (For example, the value in the length might include the size of the length field itself as well as the size of the data field, or it might be the total size of the tag, length, and data fields.)</p> <p>Enter a value from 0 to 2147483647. The parser subtracts this number from the number found in the length field to get the number of data characters that follow the length field.</p> <p>You must set this property if you have set <i>Data Element Separation</i> property to Tagged Encoded Length and the actual number of data characters is less than the value found in the length field.</p>
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the <i>Data Element Separation</i> of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.</p>

#### Message TDS properties:

Property	Type	Meaning
Message Key	String	<p>Specify an alternative unique value that identifies the message in the bit stream. This property is required if the message is embedded within another message.</p> <p><b>Note:</b> In Version 6.0 the use of Message Key has been deprecated for identifying an embedded message. You now have the option of identifying an embedded message using a Message Identity.</p>

#### Simple type TDS properties:

There are no properties to show.



### Unique TDS properties:

There are no properties to show.

### Wildcard attribute TDS properties:

There are no properties to show.

### Wildcard element TDS properties:

There are no properties to show.

## Documentation properties for all message set objects

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

### Note:

1. Key, Keyref, and Unique objects do not have documentation properties.

## Message model object properties by object

The following objects have properties that can be viewed or set:

- "Attribute group reference properties"
- "Attribute reference properties" on page 234
- "Complex type properties" on page 275
- "Element reference properties" on page 284
- "Global attribute properties" on page 334
- "Global attribute group properties" on page 357
- "Global element properties" on page 358
- "Global group properties" on page 386
- "Group reference properties" on page 391
- "Key properties" on page 394
- "Keyref properties" on page 394
- "Local attribute properties" on page 395
- "Local element properties" on page 453
- "Local group properties" on page 521
- "Message properties" on page 528
- "Simple type properties" on page 531
- "Unique properties" on page 546
- "Wildcard attribute properties" on page 546
- "Wildcard element properties" on page 547

### Attribute group reference properties

An attribute group reference can have the following properties;

- "Attribute group reference logical properties" on page 178
- "Attribute group reference CWF properties" on page 204

- “Attribute group reference XML properties” on page 210
- “Attribute group reference TDS properties” on page 218
- “Documentation properties for all message set objects” on page 175

**Attribute group reference logical properties:**

Property	Type	Meaning
Reference Name	Enumerated type	The <i>Reference Name</i> is the name of the object that this object is referring to. The objects available to reference can be selected from the drop down list.

**Attribute group reference CWF properties:**

There are no properties to show.

**Attribute group reference XML properties:**

There are no properties to show.

**Attribute group reference TDS properties:**

There are no properties to show.

**Documentation properties for all message set objects:**

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

**Note:**

1. Key, Keyref, and Unique objects do not have documentation properties.

**Attribute reference properties**

An attribute reference can have the following properties;

- “Attribute reference logical properties” on page 178
- “Attribute reference CWF properties” on page 204
- “Attribute reference XML properties” on page 210
- “Attribute reference TDS properties” on page 218
- “Documentation properties for all message set objects” on page 175

**Attribute reference logical properties:**

Property	Type	Meaning
Reference Name	Enumerated type	The <i>Reference Name</i> is the name of the object that this object is referring to. The objects available to reference can be selected from the drop down list.

Property	Type	Meaning
Usage	Enumerated type	<p>The usage property is used in conjunction with the <i>Value</i> property found in an attribute object. The default for the <i>Usage</i> property is optional.</p> <p>Select from;</p> <ul style="list-style-type: none"> <li>• optional. <ul style="list-style-type: none"> <li>– Where the <i>Value</i> property is set to default and no data has been entered in the <i>Value</i> property, the attribute can appear once and can have any value.</li> <li>– Where the <i>Value</i> property is set to default, the attribute can appear once. If it does not appear, its value will be the data that has been entered in the <i>Value</i> property. If it does appear it will be the value given.</li> <li>– Where the <i>Value</i> property is set to fixed, the attribute can appear once. If it does appear, its value must match the data that has been entered in the <i>Value</i> property. If it does not appear its value will be the data that has been entered in the <i>Value</i> property.</li> </ul> </li> <li>• prohibited. The attribute must not appear.</li> <li>• required. <ul style="list-style-type: none"> <li>– Where the <i>Value</i> property is set to default and no data has been entered in the <i>Value</i> property, the attribute must appear once and can have any value.</li> <li>– Where the <i>Value</i> property is set to fixed, the attribute must appear once and it must match the data that has been entered in the <i>Value</i> property.</li> </ul> </li> </ul>

#### Attribute reference CWF properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types	Boolean types	DateTime types	Decimal types
<ul style="list-style-type: none"> <li>- base64Binary</li> <li>- hexBinary</li> </ul>	<ul style="list-style-type: none"> <li>- boolean</li> </ul>	<ul style="list-style-type: none"> <li>- date</li> <li>- dateTime</li> <li>- gDay</li> <li>- gMonth</li> <li>- gMonthDay</li> <li>- gYear</li> <li>- gYearMonth</li> <li>- time</li> </ul>	<ul style="list-style-type: none"> <li>- decimal</li> <li>- integer</li> <li>- negativeInteger</li> <li>- nonNegativeInteger</li> <li>- nonPositiveInteger</li> <li>- positiveInteger</li> </ul>

<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>
--	---	--	--

**Note:**

1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.

*CWF properties for attribute reference and local attribute binary types:*

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Binary schema types: base64Binary, hexBinary

**Physical representation**

Property	Type	Meaning
Length Count	Button and Integer	<p>If you have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 1.</p> <p>The maximum value that you can specify is 2147483647.</p> <p>The default value is empty (not set).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>

## Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.</p> <p>For repeating objects, this property is applied to all instances.</p>

*CBF properties for attribute reference and local attribute boolean types:*

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Boolean schema types: boolean

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	Specify how the object is aligned from the start of the message. Select one of: <ul style="list-style-type: none"><li>• 1 Bytes. This is the default value.</li><li>• 2 Bytes</li><li>• 4 Bytes</li><li>• 8 Bytes</li><li>• 16 Bytes</li></ul>
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

*CBF properties for attribute reference and local attribute dateTime types:*

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>Fixed Length String. The element's length is determined by other length properties below.</li> <li>Length Encoded String 1. The element's first byte contains the length of the string following the length byte in length units. The maximum length of a Length Encoded String 1 element is 255 length units.</li> <li>Length Encoded String 2. The element's first two bytes contain the length of the string following the 2 length bytes in length units. The maximum length of a Length Encoded String 2 element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding.</li> <li>Null Terminated String. The string ends with the hexadecimal NULL character, 'X'00'.</li> <li>Packed Decimal. The date<code>Time</code> is coded as a Packed Decimal number. It is valid only if the <i>Date<code>Time</code> Format</i> property represents numeric-only data.</li> <li>Binary. The date<code>Time</code> is encoded as a binary sequence of bytes. If you select this option, the range of symbols that you can specify for the Format String property is less than the range of symbols you can specify if you select a string option (see "Date<code>Time</code> formats" on page 701 for details).</li> <li>Time Seconds. This value supports C <code>time_t</code> and Java Date and Time objects. It is valid only if the <i>Date<code>Time</code> Format</i> property represents numeric-only data.</li> <li>Time Milliseconds. This value supports C <code>time_t</code> and Java Date and Time objects. It is valid only if the <i>Date<code>Time</code> Format</i> property represents numeric-only data.</li> </ul> <p>The default value is fixed length string.</p>
Date <code>Time</code> Format	String	<p>Specify a template for date and time.</p> <p>The default date<code>Time</code> format is dependent on the logical type of the object. For information on the defaults for the date<code>Time</code> format according to the logical type see "Date<code>Time</code> defaults by logical type" on page 709.</p> <p>If you set the <i>Physical Type</i> to Binary, the template is restricted to those components defined in "Date<code>Time</code> as string data" on page 701. If you set the <i>Physical Type</i> to Packed Decimal, Time Seconds, or Time Milliseconds, the template is restricted to those components that represent numbers. In these cases, you must update this <i>Date<code>Time</code> Format</i> property.</p> <p>See "Date<code>Time</code> formats" on page 701 for details of date and time formats.</p>
Length Count	Button and Integer	<p>If you have selected a <i>Physical Type</i> of Fixed Length String, Packed Decimal, or Binary, and have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 1 for all three physical types.</p> <p>The maximum value that you can specify is 256 for Fixed Length String, 10 for Packed Decimal, and 2147483647 for Binary.</p> <p>The default value is 0 (zero).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	<p>If you have set the <i>Physical Type</i> property to Packed Decimal, Time Seconds, or Time Milliseconds, select (the default) or unselect <i>Signed</i>. If you have selected another value for <i>Physical Type</i>, this property is invalid.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to Fixed Length String, select Left Justify (the default value) or Right Justify from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>



Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Fixed Length String, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

*CWF properties for attribute reference and local attribute decimal types:*

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	Select one of the following from the drop-down list: <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer and Packed Decimal are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	Enter the number of bytes to specify the element length: <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you set the <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 10.</li> <li>• If you set the <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)</li> </ul>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	Select (the default) or deselect this property. This property is used in conjunction with <i>Sign Orientation</i> .
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to External Decimal and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• Included Leading. This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• Separate Leading. This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• Included Trailing. This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• Separate Trailing. This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
Virtual Decimal Point	Integer	<p>Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point should be moved from its assumed position. For example, a decimal element containing 1234 with a Virtual Decimal value of 3 is 1.234. This is equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

CWF properties for attribute reference and local attribute float types:

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Float schema types: double, float

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	Select one of the following from the drop-down list: <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Float. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL. This is the default value.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer, Packed Decimal, and Float are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	Enter the number of bytes to specify the element length: <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Float, select a value from the drop-down list. The default value is 8.</li> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you set the <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 10.</li> <li>• If you set the <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)</li> </ul>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	<p>Select or deselect (unsigned, the default) this property. If you have set <i>Physical Type</i> to <i>Float</i>, this is selected. This property is used in conjunction with <i>Sign Orientation</i>.</p>
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to <i>External Decimal</i> and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to <i>Leading</i> or <i>Trailing</i> (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• <b>Included Leading.</b> This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• <b>Separate Leading.</b> This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• <b>Included Trailing.</b> This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• <b>Separate Trailing.</b> This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
Virtual Decimal Point	Integer	<p>Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point should be moved from its assumed position. For example, a float element containing 1234 with a Virtual Decimal value of 3 is 1.234.</p> <p>This is not applicable if you have set <i>Physical Type</i> to Float.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>



Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

*CWF properties for attribute reference and local attribute integer types:*

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedLong, unsignedShort

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	Select one of the following from the drop-down list: <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer and Packed Decimal are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	Enter the number of bytes to specify the element length: <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you have set <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 6.</li> <li>• If you have set <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 11.</li> </ul>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	Select (the default) or deselect this property. This property is used in conjunction with <i>Sign Orientation</i> .
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to External Decimal and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• <b>Included Leading.</b> This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• <b>Separate Leading.</b> This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• <b>Included Trailing.</b> This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• <b>Separate Trailing.</b> This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

*CWF properties for attribute reference and local attribute string types:*

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	Select one of the following from the drop-down list: <ul style="list-style-type: none"> <li>• Fixed Length String. The element's length is determined by other length properties below.</li> <li>• Length Encoded String 1. The element's first byte contains the length of the string following the length byte in length units. The maximum length of a Length Encoded String 1 element is 255 length units.</li> <li>• Length Encoded String 2. The element's first two bytes contain the length of the string following the 2 length bytes in length units. The maximum length of a Length Encoded String 2 element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding.</li> <li>• Null Terminated String. The string ends with the hexadecimal NULL character, X'00'.</li> </ul> <p>The default is Fixed Length String.</p>
Length Count	Button and Integer	If you have selected a <i>Physical Type</i> of Fixed Length String or Binary, and have set <i>Length Type</i> to Count, enter the number of length units for the element.  The minimum value that you can specify is 0 (zero), the maximum value that you can specify is 2147483647  The default value is 0 (zero).

Property	Type	Meaning
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message’s CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message’s CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to Fixed Length String, select Left Justify (the default value) or Right Justify from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Fixed Length String, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>



Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

### Attribute reference XML properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
Float types - double - float	Integer types - byte - int - long - short - unsignedByte - unsignedInt - unsignedLong - unsignedShort	Interval types - duration <sup>1</sup>	String types - anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

**Note:**

1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.

*XML properties for attribute reference, element reference, local attribute, local element binary types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Binary schema types: base64Binary, hexBinary

### **Field identification**

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

## Physical representation

Property	Type	Meaning
Encoding	String	<p>Select one of the following values from the drop-down list: :</p> <ul style="list-style-type: none"> <li>• CDatahex (the default)</li> <li>• hex</li> <li>• base64</li> </ul>

*XML properties for attribute reference, element reference, local attribute, local element boolean types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Boolean schema types: boolean

## Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element dateTime types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

## Physical representation

Property	Type	Meaning
DateTime Format	String	<p>Specify a format string that specifies the rendering of the value for <code>dateTime</code> elements.</p> <p>The default <code>dateTime</code> format is dependent on the logical type of the object. For information on the defaults for the <code>dateTime</code> format according to the logical type see "DateTime defaults by logical type" on page 709.</p> <p>See "DateTime formats" on page 701 for details of <code>dateTime</code> formats.</p>

*XML properties for attribute reference, element reference, local attribute, local element decimal types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Decimal schema types: `decimal`, `integer`, `negativeInteger`, `nonNegativeInteger`, `nonPositiveInteger`, `positiveInteger`



## Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li> <p><b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element.</p> <p>If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</p> </li> <li> <p><b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements.</p> <p>If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</p> </li> <li> <p><b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name</i>, <i>ID Attribute Name</i>, <i>ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li> <p><b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</p> </li> <li> <p><b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name</i>, <i>ID Attribute Name</i>, <i>ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element float types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Float schema types: double, float

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element integer types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Integer schema types: `byte`, `int`, `long`, `short`, `unsignedByte`, `unsignedInt`, `unsignedLong`, `unsignedShort`

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element string types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is <i>id</i>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <i>val</i>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

### Attribute reference TDS properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
---	----------------------------	--	---



<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>
<p><b>Note:</b></p> <p>1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.</p>			

*TDS properties for attribute reference binary types:*

The TDS Format properties described here apply to:

- Objects: Attribute Reference
- Binary schema types: base64Binary, hexBinary

Property	Type	Meaning
Length Reference	Enumerated type	Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.  For information about reordering elements, see "Reordering objects" on page 106.

*TDS properties for attribute reference boolean types:*

The TDS Format properties described here apply to:

- Objects: Attribute Reference
- Boolean schema types: boolean

Property	Type	Meaning
Length Reference	Enumerated type	Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.  For information about reordering elements, see "Reordering objects" on page 106.

*TDS properties for attribute reference dateTime types:*

The TDS Format properties described here apply to:

- Objects: Attribute Reference

- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Property	Type	Meaning
Length Reference	Enumerated type	Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.  For information about reordering elements, see "Reordering objects" on page 106.

*TDS properties for attribute reference decimal types:*

The TDS Format properties described here apply to:

- Objects: Attribute Reference
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger

Property	Type	Meaning
Length Reference	Enumerated type	Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.  For information about reordering elements, see "Reordering objects" on page 106.

*TDS properties for attribute reference float types:*

The TDS Format properties described here apply to:

- Objects: Attribute Reference
- Float schema types: double, float

Property	Type	Meaning
Length Reference	Enumerated type	Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.  For information about reordering elements, see "Reordering objects" on page 106.

*TDS properties for attribute reference integer types:*

The TDS Format properties described here apply to:

- Objects: Attribute Reference
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedLong, unsignedShort

Property	Type	Meaning
Length Reference	Enumerated type	Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.  For information about reordering elements, see "Reordering objects" on page 106.

*TDS properties for attribute reference string types:*

The TDS Format properties described here apply to:

- Objects: Attribute Reference
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Property	Type	Meaning
Length Reference	Enumerated type	Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.  For information about reordering elements, see “Reordering objects” on page 106.

### **Documentation properties for all message set objects:**

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### **Note:**

1. Key, Keyref, and Unique objects do not have documentation properties.

### **Complex type properties**

A complex type can have the following properties;

- “Complex type logical properties” on page 179
- “Complex type CWF properties” on page 205
- “Complex type XML properties” on page 211
- “Complex type TDS properties” on page 219
- “Documentation properties for all message set objects” on page 175

### **Complex type logical properties:**

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"> <li>• - the hyphen</li> <li>• _ the underscore</li> <li>• . the period</li> </ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <i>xml</i> or any variant (for example <i>Xml</i>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>
Base Type	Enumerated type	You can use this property to select a type (simple or complex) that is used as the starting point to define a new complex type that is derived by restriction or extension.
Derived By	Enumerated type	<p>If this property is active, select from;</p> <ul style="list-style-type: none"> <li>• <i>restriction</i>. If a complex type is derived by restriction, the content model of the complex type is a subset of the base type.</li> <li>• <i>extension</i>. If the complex type is derived by extension, the content model of the complex type is the content model of the base type plus the content model specified in the type derivation.</li> </ul> <p>Derivation by list or union is not supported.</p>

## Content

The table below shows the valid settings for *Composition* and *Content Validation*. These properties are actually located on the group which defines the content of this type. They can only be edited if the *Local group* button is selected. If the *Global group* button is selected, these properties are taken from the global group identified by the *Group name* field.

Valid children in a complex type that depend on both *Composition* and *Content Validation* are shown in "Content Validation properties for complex types" on page 182.

Property	Type	Meaning
Local Group	Button	You should select this if the content of your complex type is a local group.

Property	Type	Meaning
Composition	Enumerated type	<p>The property describes how the message tree is structured and is used in combination with the property <i>Content Validation</i>.</p> <p><i>Composition</i> determines, for example, if the elements within the tree can appear in any order, or if the order is predefined.</p> <p>If you set this property to Ordered Set or Sequence, the order of elements in the input message when the message is parsed, and the order in the logical tree when the output message is constructed by the parser, is important. If the order is not correct, the parser might generate an error, or might produce unexpected results. Therefore you must take care to include ESQLE SET statements in the correct order when you create a message in a Compute node.</p> <p>Select from:</p> <ul style="list-style-type: none"> <li>• Empty</li> <li>• sequence. If you select this option, you can only define children that are simple types, complex types, elements, or groups. These children, if present, must appear in the specified order. They can repeat and can be duplicated.</li> <li>• choice. If you select this option, you can define children that are simple types, complex types, or elements. Only one of the defined children of the complex type can be present, but repeating children are allowed. Use this option if you want to model C unions and COBOL REDEFINES in a Custom Wire Format, or an XML DTD element that uses choice in an XML Wire Format, or some industry standard tagged/delimited messages (for example SWIFT) use this format.</li> <li>• all. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can only contain elements - groups are not allowed. An all group can only be used at the top level of a complex type - it cannot be a member of another group within a type.</li> <li>• unorderedSet. If you select this option, you can define only elements as children. The elements can repeat but cannot be duplicated. Child elements can appear in any order.</li> <li>• orderedSet. If you select this option, you can define only elements as children. These elements, if present, must appear in the specified order, and they can repeat but cannot be duplicated. This is the default value for new complex types.</li> <li>• message. If you select this option, you can define only messages as children. They can repeat, but they cannot be duplicated. Like Choice, only one of the defined children can be present. If the complex type includes more than one message, the bit stream contains the exact length of the embedded message, and is not padded to the length of the longest. Use this option to model multipart messages, which are used in some industry standards, for example, SWIFT. For more information, see the section on multipart messages in "Multipart messages" on page 28.</li> </ul>

Property	Type	Meaning
Content Validation	Enumerated type	<p><i>Content Validation</i> controls how the broker responds to undeclared content and specifies where the objects that are included within the complex type are defined, if at all. It is used in combination with the <i>Composition</i> property.</p> <p>Options:</p> <ul style="list-style-type: none"> <li>• Closed. The complex type can only contain the child elements that you have added to it.</li> <li>• Open Defined. The complex type can contain any valid element defined within the message set.</li> <li>• Open. The complex type can contain any valid element, not just those that you have added to this complex type.</li> </ul> <p>See “Combinations of Composition and Content Validation” on page 280 for further details of these options.</p>
Group Reference	Button	You should select this if the content of your complex type is a reference to a group object
Group Name	Enumerated type	The <i>Group Name</i> is the name of the group that this complex type is referring to. The groups available to be referenced can be selected from the drop down list.
Min Occurs	Integer	<p>Specify the minimum number of times that the object can repeat. The default is 1.</p> <p>If the value is set to 0, then the object is optional.</p> <p>With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i>, it must be less than or equal to the value in <i>Max Occurs</i>.</p>
Max Occurs	Integer	<p>Specify the maximum number of times that the object can repeat. The default is 1.</p> <p>If this property is not set, then the object can not occur more than once.</p> <p>If this property is set to 0, it will be interpreted as if the object does not exist in the message.</p> <p>It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.</p>
Mixed	Check box	Select this where the complex type has mixed content and contains character data alongside sub-elements.

### Substitution settings

Property	Type	Meaning
Final	Multiple selection enumerated type	<p>The final attribute on a complex type controls whether other types may be derived from it. Valid values are extension/restriction/all. You can select from one or more of the following:</p> <ul style="list-style-type: none"> <li>• Empty</li> <li>• restriction. Prohibit type substitution by elements whose types are restrictions of the head element’s type.</li> <li>• extension. Prohibit type substitution by elements whose types are extensions of the head element’s type.</li> <li>• all. Prohibit substitution by any method.</li> </ul> <p>To select more than one, you will need to type the selection into the property field.</p>

Property	Type	Meaning
Block	Multiple selection enumerated type	<p>The block attribute on a complex type restricts the types of substitutions which are allowed for elements based on that type. In the WebSphere Message Broker its effect is the same as if the block attribute were copied from the complex type onto every element based on the complex type. You can select from one or more of the following:</p> <ul style="list-style-type: none"> <li>• Empty</li> <li>• restriction. Prohibit type substitution by elements whose types are restrictions of the head element's type.</li> <li>• extension. Prohibit type substitution by elements whose types are extensions of the head element's type.</li> <li>• all. Prohibit substitution by any method.</li> </ul> <p>To select more than one, you will need to type the selection into the property field.</p>
Abstract	Check box	If selected, no elements based on this type can appear in the message.

*Content Validation properties for complex types:*

*Content Validation* specifies where the objects that are included within a complex type or group are defined, if at all. It is used in combination with *Composition*.

The first table below shows the valid settings for *Content Validation* if *Composition* is set to Message, and the second table shows the valid settings for *Content Validation* if *Composition* is not set to Message.

### Content Validation options if Composition is set to Message

Option	Meaning
Open	When a message is parsed, this complex type or group can contain any message, not just those that you have defined in this message set. You can use this option for sparse messages (see "Predefined and self-defining elements and messages" on page 32 for a definition of sparse messages).
Closed	When a message is parsed, this complex type or group can only contain the messages that are members of this complex type or group. This is always the case for messages represented in CWF format.
Open Defined	When a message is parsed, this complex type or group can contain any message defined within the message set.

### Content Validation options if Composition is not set to Message

Option	Meaning
Open	When a message is parsed, this complex type or group can contain any elements and not just those that you have defined in this message set (see "Predefined and self-defining elements and messages" on page 32 for a definition of sparse messages).
Closed	When a message is parsed, this complex type or group can only contain the elements that are members of this complex type or group.
Open Defined	When a message is parsed, this complex type or group can contain any element that you have defined within the message set.

When you are using Content Validation set to open or open defined, you can not specify the precise position where the content that is not modeled is permitted to

occur. If you wish to do this, you should consider using a wildcard element as an alternative. Note that wildcard elements can only appear within a complex type or group with Composition of sequence and Content Validation of closed.

*Combinations of Composition and Content Validation:*

The table below shows how the choices that you make are affected by the settings you choose for the Composition and Content Validation properties.

### **Valid children in complex types dependent on Composition and Content Validation**

Composition	Content Validation	Valid children
Empty	Closed	None
Empty	Open	None
Empty	Open Defined	None
Sequence	Open	Elements, group references, embedded simple types
Sequence	Closed	Elements, group references, embedded simple types
Sequence	Open Defined	Elements, group references, embedded simple types
Choice	Closed	Elements, group references, embedded simple types
All	Closed	All children are valid
All	Open	All children are valid
All	Open Defined	All children are valid
Unordered Set	Open	Elements
Unordered Set	Closed	Elements
Unordered Set	Open Defined	Elements
Ordered Set	Open	Elements
Ordered Set	Closed	Elements
Ordered Set	Open Defined	Elements
Message	Open	Messages
Message	Closed	Messages
Message	Open Defined	Messages

*Valid combinations of repeat and duplicate elements in complex types:*

The table below defines the valid combinations of repeated and duplicate elements within a complex type, dependent on the Composition property value.

- A repeated element is an element that is included once within the complex type, and is defined with the property Min Occurs set to greater than 1. Repeated elements are therefore always contiguous and are always specified in the form A[n].
- A duplicate element is an element included more than once anywhere within the complex type. Duplicate elements do not have to be contiguous.



## Repeated and duplicate elements in a complex type

Elements in type	Example	Unordered Set	Ordered Set	Sequence
No repeats, no duplicates	A, B, C	Yes	Yes	Yes
Repeated element (contiguous)	A[n], B, C	Yes	Yes	Yes
Duplicate element A (contiguous)	A, A, B, C	No	No	Yes
Duplicate element A (non-contiguous)	A, B, C, A	No	No	Yes

### Complex type CWF properties:

There are no properties to show.

### Complex type XML properties:

There are no properties to show.

### Complex type TDS properties:

#### Field Identification

If the complex type is based on a global group, the TDS properties listed below will actually be located on the global group. If this is the case, any changes to these properties will be applied to the global group, and will affect all references to the group (including any other complex types which are based on it).

Property	Type	Meaning
Data Element Separation	Enumerated Type	<p>Specify the method used to separate the data elements within the type. Select one of the following values:</p> <ul style="list-style-type: none"> <li>• Tagged Delimited. This value indicates that all elements within the complex type are identified by a tag, and separated by the value specified in the optional <i>Delimiter</i> property (if specified). You must set the <i>Tag</i> property for all child elements of simple type, and you may set the <i>Delimiter</i> property to a non-empty value. See “Global element TDS properties” on page 223. You must also set the <i>Tag Data Separator</i> or the <i>Length of Tag</i> properties.</li> <li>• Tagged Fixed Length. This value indicates that each element is identified by a tag, and the data has a fixed length. There are no delimiters. You must set the <i>Tag</i> property for each of the child elements of this complex type, and each child element must have a <i>Length</i> or <i>Length Reference</i> property assigned to it. You must also set the <i>Tag Data Separator</i> or the <i>Length of Tag</i> properties.</li> <li>• Tagged Encoded Length. This value indicates that all elements within the complex type are separated by a tag, and a length field follows each tag. There are no delimiters. The tag can be fixed length, as set by <i>Length of Tag</i>, or variable length delimited by the <i>Tag Data Separator</i>. You must also set <i>Length Of Encoded Length</i> so that the parser knows the size of the length field, and set <i>Extra Chars in Encoded Length</i> to tell the parser how many to subtract from the value in <i>Length Of Encoded Length</i> to get the actual length of data that follows the length field.</li> </ul> <p>This method provides a more flexible way of handling ACORD AL3 standard messages than Fixed Length AL3, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard.</p> <ul style="list-style-type: none"> <li>• All Elements Delimited. This value indicates that all elements within the complex type are separated by a delimiter. You must set the value in the <i>Delimiter</i> property.</li> <li>• Variable Length Elements Delimited. This value indicates that some of the elements within the complex type might be of variable length: if they are, they must be delimited by the value specified in the <i>Delimiter</i> property.</li> <li>• Use Data Pattern. This value indicates that the parser determines the elements by matching the data with the regular expression set in the element or type member <i>Data Pattern</i> property. See “Message definition file properties” on page 174.</li> <li>• Fixed Length. This value indicates that all elements within the complex type are fixed length. The next data element is accessed by adding the value of the <i>Length</i> property to the offset (see “Global element TDS properties” on page 223). If you set the <i>Data Element Separation</i> property of a complex type to Fixed Length, you must also set the <i>Data Element Separation</i> property of all complex children of this type to Fixed Length. Each child element must have a <i>Length</i> or <i>Length Reference</i> property assigned to it.</li> <li>• Fixed Length AL3. This value has a similar meaning to the separation type Fixed Length, but also indicates to the parser that a number of predefined rules with regard to missing optional elements, encoded lengths, and versioning must be applied. If you set the <i>Data Element Separation</i> property of a complex type to Fixed Length AL3, you must also set the <i>Data Element Separation</i> property of all complex children of this type to Fixed Length AL3.</li> <li>• Undefined. This value is set automatically if you set the <i>Type Composition</i> property of a complex type to Message, and you cannot change it to any other value.</li> </ul> <p>Do not set the <i>Type Composition</i> property to Empty, Choice, Unordered Set, Ordered Set, Sequence, or Simple Unordered Set. If you do, you will be unable to check in the type.</p>
Group Indicator	String	Specify the value of a special character or string that precedes the data belonging to a group or complex type within the bit stream.

Property	Type	Meaning
Group Terminator	String	Specify the value of a special character or string that terminates data belonging to a group or a complex type within the bit stream.
Delimiter	String	Specify the value of a special character or string that specifies the delimiter used between data elements.  This property applies only to the delimited <i>Data Element Separation</i> methods (Tagged Delimited, All Elements Delimited, and Variable Elements Delimited).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select if you want delimiters to be suppressed for elements that are missing within a message. Select from: <ul style="list-style-type: none"> <li>• End Of Type. Use this option to suppress the delimiter when an element is missing. For example, if the model has been defined to have up to 3 elements and only 2 are present, the last delimiter can be omitted from the message.</li> <li>• Never. Use this option to ensure that even if optional elements are not present, all delimiters will be written out. This option should be used when the delimiter used to delimit parent and child objects is the same. For example, if an optional child element is missing, message processing applications could not tell where the child elements in a message ended and the next parent element started if the delimiters are all the same.</li> </ul>
Observe Element Length	Check box	Applicable when <i>Data Element Separation</i> is All Elements Delimited and tells the TDS parser to take any <i>Length</i> property of child elements or attributes into account. The default value depends on the setting of the <i>Messaging Standard</i> property (at the message set level) and <i>Data Element Separation</i> properties. <ul style="list-style-type: none"> <li>• Where <i>Data Element Separation</i> is All Elements Delimited and the <i>Messaging Standard</i> is set to TLOG , this property should be set. For all other messaging standards it should not be set.</li> <li>• Where <i>Data Element Separation</i> is Tagged Delimited this property should not be set.</li> <li>• Where <i>Data Element Separation</i> is Tagged Fixed Length, Fixed Length, Fixed Length AL3, or Variable Length Elements Delimited this property will be set and is disabled.</li> <li>• For all other data element separation methods, this property is not set and is disabled.</li> </ul> Any other combination will generate a task list warning.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The <i>Tag Data Separator</i> and <i>Length of Tag</i> properties are mutually exclusive.  If you set the property <i>Tag Data Separator</i> , it overrides <i>Length of Tag</i> .  This property applies only to the tagged <i>Data Element Separation</i> methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).
Length of Tag	Integer	Specify the length of a tag value. When the message is parsed, this allows tags to be extracted from the bit stream if the <i>Tag Data Separator</i> property is not set.  The <i>Tag Data Separator</i> and <i>Length of Tag</i> properties are mutually exclusive. If you set the property <i>Tag Data Separator</i> , it overrides this value.  This property applies only to the tagged <i>Data Element Separation</i> methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).

Property	Type	Meaning
Length of Encoded Length	Integer	<p>Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647.</p> <p>You must set this property if you have set <i>Data Element Separation</i> property to Tagged Encoded Length; it is not valid otherwise.</p> <p>The actual number of data characters parsed also depends on the value of the <i>Extra Chars in Encoded Length</i> property.</p>
Extra Chars in Encoded Length	Integer	<p>(Only valid if the <i>Data Element Separation</i> method is set to Tagged Encoded Length.) Specifies the number of extra characters included in the value found in the length field. (For example, the value in the length might include the size of the length field itself as well as the size of the data field, or it might be the total size of the tag, length, and data fields.)</p> <p>Enter a value from 0 to 2147483647. The parser subtracts this number from the number found in the length field to get the number of data characters that follow the length field.</p> <p>You must set this property if you have set <i>Data Element Separation</i> property to Tagged Encoded Length and the actual number of data characters is less than the value found in the length field.</p>

#### Documentation properties for all message set objects:

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### Note:

1. Key, Keyref, and Unique objects do not have documentation properties.

#### Element reference properties

An element reference can have the following properties:

- “Element reference logical properties” on page 183
- “Element reference CWF properties” on page 205
- “Element reference XML properties” on page 211
- “Element reference TDS properties” on page 222
- “Documentation properties for all message set objects” on page 175

#### Element reference logical properties:

Property	Type	Meaning
Reference Name	Enumerated type	The <i>Reference Name</i> is the name of the object that this object is referring to. The objects available to reference can be selected from the drop down list.

## Occurrences

Property	Type	Meaning
Min Occurs	Integer	<p>Specify the minimum number of times that the object can repeat. The default is 1.</p> <p>If the value is set to 0, then the object is optional.</p> <p>With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i>, it must be less than or equal to the value in <i>Max Occurs</i>.</p>
Max Occurs	Integer	<p>Specify the maximum number of times that the object can repeat. The default is 1.</p> <p>If this property is not set, then the object can not occur more than once.</p> <p>If this property is set to 0, it will be interpreted as if the object does not exist in the message.</p> <p>It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.</p>

The Min Occurs and Max Occurs properties are used in conjunction with an elements Value properties. The table below summarizes how an element reference can be constrained.

Min Occurs	Max Occurs	Fixed	Default	Notes
1	1			The element must appear once and can have any value.
1	1	Delta		The element must appear once and it must match the data that has been entered in the <i>Value</i> property. In this example the element must contain the text Del ta.
2	-1	Delta		The element must appear twice or more and it must match the data that has been entered in the <i>Value</i> property. In this example there will be at least two elements that must contain the text Del ta.
0	1			The element is optional and can appear once and have any value.
0	1	Delta		The element is optional and can appear once. If it does appear, its value must match the data that has been entered in the <i>Value</i> property. If it does not appear its value will be the data that has been entered in the <i>Value</i> property.
0	1		Delta	The element is optional and can appear once. If it does not appear, its value will be the data that has been entered in the <i>Value</i> property. If it does appear it must be the value given in the element.
0	2		Delta	The element is optional and can appear once, twice or not at all. If the element does not appear it is not provided. If the element appears and it is empty, it set to the data held in the <i>Value</i> property, else it is the value given in the element.

Min Occurs	Max Occurs	Fixed	Default	Notes
0	0			The element is prohibited and must not appear.

### Element reference CWF properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
Float types - double - float	Integer types - byte - int - long - short - unsignedByte - unsignedInt - unsignedLong - unsignedShort	Interval types - duration <sup>1</sup>	String types - anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

**Note:**

1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.

### *CWF properties for element reference and local element binary types:*

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Binary schema types: base64Binary, hexBinary

## Physical representation

Property	Type	Meaning
Length Count	Button and Integer	<p>If you have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 1.</p> <p>The maximum value that you can specify is 2147483647.</p> <p>The default value is empty (not set).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message’s CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message’s CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>

## Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.  Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.  If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.

*CWF properties for element reference and local element boolean types:*

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Boolean schema types: boolean

## Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	Specify how the object is aligned from the start of the message. Select one of: <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>



Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

### Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.  Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.  If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.

*CWF properties for element reference and local element dateTime types:*

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Fixed Length String. The element's length is determined by other length properties below.</li> <li>• Length Encoded String 1. The element's first byte contains the length of the string following the length byte in length units. The maximum length of a Length Encoded String 1 element is 255 length units.</li> <li>• Length Encoded String 2. The element's first two bytes contain the length of the string following the 2 length bytes in length units. The maximum length of a Length Encoded String 2 element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding.</li> <li>• Null Terminated String. The string ends with the hexadecimal NULL character, 'X'00'.</li> <li>• Packed Decimal. The date<code>Time</code> is coded as a Packed Decimal number. It is valid only if the <i>Date<code>Time</code> Format</i> property represents numeric-only data.</li> <li>• Binary. The date<code>Time</code> is encoded as a binary sequence of bytes. If you select this option, the range of symbols that you can specify for the Format String property is less than the range of symbols you can specify if you select a string option (see "Date<code>Time</code> formats" on page 701 for details).</li> <li>• Time Seconds. This value supports C <code>time_t</code> and Java Date and Time objects. It is valid only if the <i>Date<code>Time</code> Format</i> property represents numeric-only data.</li> <li>• Time Milliseconds. This value supports C <code>time_t</code> and Java Date and Time objects. It is valid only if the <i>Date<code>Time</code> Format</i> property represents numeric-only data.</li> </ul> <p>The default value is fixed length string.</p>
Date <code>Time</code> Format	String	<p>Specify a template for date and time.</p> <p>The default date<code>Time</code> format is dependent on the logical type of the object. For information on the defaults for the date<code>Time</code> format according to the logical type see "Date<code>Time</code> defaults by logical type" on page 709.</p> <p>If you set the <i>Physical Type</i> to Binary, the template is restricted to those components defined in "Date<code>Time</code> as string data" on page 701. If you set the <i>Physical Type</i> to Packed Decimal, Time Seconds, or Time Milliseconds, the template is restricted to those components that represent numbers. In these cases, you must update this <i>Date<code>Time</code> Format</i> property.</p> <p>See "Date<code>Time</code> formats" on page 701 for details of date and time formats.</p>
Length Count	Button and Integer	<p>If you have selected a <i>Physical Type</i> of Fixed Length String, Packed Decimal, or Binary, and have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 1 for all three physical types.</p> <p>The maximum value that you can specify is 256 for Fixed Length String, 10 for Packed Decimal, and 2147483647 for Binary.</p> <p>The default value is 0 (zero).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	<p>If you have set the <i>Physical Type</i> property to Packed Decimal, Time Seconds, or Time Milliseconds, select (the default) or unselect <i>Signed</i>. If you have selected another value for <i>Physical Type</i>, this property is invalid.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to Fixed Length String, select Left Justify (the default value) or Right Justify from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Fixed Length String, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>• <b>NULLPadFill</b>. This is only valid if <i>Physical Type</i> is Fixed Length String. The field is filled with the value specified by the <i>Padding Character</i>. This is the default value.</li> <li>• <b>NULLLogicalValue</b>. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>• <b>NULLLiteralValue</b>. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. Use this option when the value you have set for <i>Encoding Null Value</i> to specify a null date is not a date<code>Time</code> value, or does not conform to the standard date<code>Time</code> format <code>yyyy-MM-dd 'T'HH:mm:ss</code>.</li> <li>• <b>NULLLiteralFill</b>. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>
Encoding Null Value	String	<p>If you set the <i>Encoding Null</i> property to <b>NULLPadFill</b>, this property is disabled (grayed out).</p> <p>If you set the <i>Encoding Null</i> property to <b>NULLLogicalValue</b>, you must set this property to an ISO8601 date<code>Time</code> format. These formats are described in “Date<code>Time</code> as string data” on page 701. For example, specify a value conforming to <code>yyyy-MM-dd'T'HH:mm:ss</code> such as 1970-12-01.</p> <p>If you set the <i>Encoding Null</i> property to <b>NULLLiteralValue</b>, you can enter any value that is the same length as the field.</p> <p>If you set the <i>Encoding Null</i> property to <b>NULLLiteralFill</b>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <b>SPACE</b>, <b>NUL</b>, <b>0x00</b> or <b>0xFF</b> from the drop-down list</li> <li>• Enter a character between quotation marks, for example 'c' or "c", where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where YY is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where YY is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where xxxx is a Unicode value specified in hexadecimal format.</li> </ul>

## Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.  Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.  If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.

*CWF properties for element reference and local element decimal types:*

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>Elements that have <i>Physical Type</i> set to Integer and Packed Decimal are represented in the appropriate WebSphere MQ Encoding value.</li> <li>Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	<p>Enter the number of bytes to specify the element length:</p> <ul style="list-style-type: none"> <li>If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>If you set the <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 10.</li> <li>If you set the <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)</li> </ul>
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>Bytes.</li> <li>Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	<p>Select (the default) or deselect this property. This property is used in conjunction with <i>Sign Orientation</i>.</p>

Property	Type	Meaning
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to External Decimal and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• Included Leading. This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• Separate Leading. This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• Included Trailing. This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• Separate Trailing. This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
Virtual Decimal Point	Integer	<p>Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point should be moved from its assumed position. For example, a decimal element containing 1234 with a Virtual Decimal value of 3 is 1.234. This is equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>



Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is External Decimal. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string. You can specify a nonnumeric value for <i>Encoding Null Value</i>.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>

Property	Type	Meaning
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above, with the exception of <code>NULLLiteralFill</code>. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <i>c</i> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <i>YY</i> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <i>YY</i> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <i>xxxx</i> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

*CWF properties for element reference and local element float types:*

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Float schema types: double, float

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Float. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL. This is the default value.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer, Packed Decimal, and Float are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	<p>Enter the number of bytes to specify the element length:</p> <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Float, select a value from the drop-down list. The default value is 8.</li> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you set the <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 10.</li> <li>• If you set the <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)</li> </ul>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	<p>Select or deselect (unsigned, the default) this property. If you have set <i>Physical Type</i> to <i>Float</i>, this is selected. This property is used in conjunction with <i>Sign Orientation</i>.</p>
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to <i>External Decimal</i> and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to <i>Leading</i> or <i>Trailing</i> (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• <b>Included Leading.</b> This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• <b>Separate Leading.</b> This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• <b>Included Trailing.</b> This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• <b>Separate Trailing.</b> This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
Virtual Decimal Point	Integer	<p>Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point should be moved from its assumed position. For example, a float element containing 1234 with a Virtual Decimal value of 3 is 1.234.</p> <p>This is not applicable if you have set <i>Physical Type</i> to Float.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is External Decimal. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string. You can specify a nonnumeric value for <i>Encoding Null Value</i>.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>

Property	Type	Meaning
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above, with the exception of <code>NULLLiteralFill</code>. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <code>c</code> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <code>YY</code> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <code>YY</code> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <code>xxxx</code> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

*CWF properties for element reference and local element integer types:*

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedLong, unsignedShort

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer and Packed Decimal are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	<p>Enter the number of bytes to specify the element length:</p> <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you have set <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 6.</li> <li>• If you have set <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 11.</li> </ul>



Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	Select (the default) or deselect this property. This property is used in conjunction with <i>Sign Orientation</i> .
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to External Decimal and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• <b>Included Leading.</b> This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• <b>Separate Leading.</b> This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• <b>Included Trailing.</b> This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• <b>Separate Trailing.</b> This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is External Decimal. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string. You can specify a nonnumeric value for <i>Encoding Null Value</i>.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>

Property	Type	Meaning
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above, with the exception of <code>NULLLiteralFill</code>. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <i>c</i> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <i>YY</i> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <i>YY</i> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <i>xxxx</i> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

*CWF properties for element reference and local element string types:*

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Fixed Length String. The element's length is determined by other length properties below.</li> <li>• Length Encoded String 1. The element's first byte contains the length of the string following the length byte in length units. The maximum length of a Length Encoded String 1 element is 255 length units.</li> <li>• Length Encoded String 2. The element's first two bytes contain the length of the string following the 2 length bytes in length units. The maximum length of a Length Encoded String 2 element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding.</li> <li>• Null Terminated String. The string ends with the hexadecimal NULL character, X'00'.</li> </ul> <p>The default is Fixed Length String.</p>
Length Count	Button and Integer	<p>If you have selected a <i>Physical Type</i> of Fixed Length String or Binary, and have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 0 (zero), the maximum value that you can specify is 2147483647</p> <p>The default value is 0 (zero).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to Fixed Length String, select Left Justify (the default value) or Right Justify from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Fixed Length String, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is Fixed Length String. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>

Property	Type	Meaning
Encoding Null Value	STRING	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. If specified, its length must be equal to the length of the string element, with the exception of <code>NULLLiteralFill</code>.</p> <p>The default value is empty (not set).</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <code>c</code> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <code>YY</code> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <code>YY</code> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <code>xxxx</code> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>



## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

### Element reference XML properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
Float types - double - float	Integer types - byte - int - long - short - unsignedByte - unsignedInt - unsignedLong - unsignedShort	Interval types - duration <sup>1</sup>	String types - anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token
<b>Note:</b> 1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.			

*XML properties for attribute reference, element reference, local attribute, local element binary types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Binary schema types: base64Binary, hexBinary

### **Field identification**

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

## Physical representation

Property	Type	Meaning
Encoding	String	<p>Select one of the following values from the drop-down list: :</p> <ul style="list-style-type: none"> <li>• CDatahex (the default)</li> <li>• hex</li> <li>• base64</li> </ul>

*XML properties for attribute reference, element reference, local attribute, local element boolean types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Boolean schema types: boolean

## Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element dateTime types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

## Physical representation

Property	Type	Meaning
DateTime Format	String	<p>Specify a format string that specifies the rendering of the value for <code>dateTime</code> elements.</p> <p>The default <code>dateTime</code> format is dependent on the logical type of the object. For information on the defaults for the <code>dateTime</code> format according to the logical type see "DateTime defaults by logical type" on page 709.</p> <p>See "DateTime formats" on page 701 for details of <code>dateTime</code> formats.</p>

*XML properties for attribute reference, element reference, local attribute, local element decimal types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Decimal schema types: `decimal`, `integer`, `negativeInteger`, `nonNegativeInteger`, `nonPositiveInteger`, `positiveInteger`



## Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li> <p><b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element.</p> <p>If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</p> </li> <li> <p><b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements.</p> <p>If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</p> </li> <li> <p><b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name</i>, <i>ID Attribute Name</i>, <i>ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li> <p><b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</p> </li> <li> <p><b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name</i>, <i>ID Attribute Name</i>, <i>ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element float types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Float schema types: double, float

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element integer types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Integer schema types: `byte`, `int`, `long`, `short`, `unsignedByte`, `unsignedInt`, `unsignedLong`, `unsignedShort`

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element string types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

### Element reference TDS properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
---	----------------------------	--	---



<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>
<p><b>Note:</b></p> <p>1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.</p>			

*TDS properties for element reference binary types:*

The TDS Format properties described here apply to:

- Objects: Element Reference
- Binary schema types: base64Binary, hexBinary

### Field identification

Property	Type	Meaning
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

### Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>

*TDS properties for element reference boolean types:*

The TDS Format properties described here apply to:

- Objects: Element Reference
- Boolean schema types: boolean

### Field identification

Property	Type	Meaning
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

### Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>

*TDS properties for element reference dateTime types:*

The TDS Format properties described here apply to:

- Objects: Element Reference
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

### Field identification

Property	Type	Meaning
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

## Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.  For information about reordering elements, see "Reordering objects" on page 106.

*TDS properties for element reference decimal types:*

The TDS Format properties described here apply to:

- Objects: Element Reference
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger

## Field identification

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to be used between repeating elements.  This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.  A default value is provided if the above conditions are true as follows: <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> If none of the above rules are met, a default will not be applied.

## Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.  For information about reordering elements, see "Reordering objects" on page 106.

*TDS properties for element reference float types:*

The TDS Format properties described here apply to:

- Objects: Element Reference
- Float schema types: double, float

## Field identification

Property	Type	Meaning
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

## Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>

*TDS properties for element reference integer types:*

The TDS Format properties described here apply to:

- Objects: Element Reference
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedLong, unsignedShort

## Field identification

Property	Type	Meaning
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

## Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.  For information about reordering elements, see "Reordering objects" on page 106.

*TDS properties for element reference string types:*

The TDS Format properties described here apply to:

- Objects: Element Reference
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

## Field identification

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to be used between repeating elements.  This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.  A default value is provided if the above conditions are true as follows: <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> If none of the above rules are met, a default will not be applied.

## Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.  For information about reordering elements, see "Reordering objects" on page 106.

### Documentation properties for all message set objects:

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### Note:

1. Key, Keyref, and Unique objects do not have documentation properties.

## Global attribute properties

A global attribute can have the following properties;

- “Global attribute logical properties” on page 184
- “Global attribute CWF properties” on page 206
- “Global attribute XML properties” on page 212
- “Global attribute TDS properties” on page 223
- “Documentation properties for all message set objects” on page 175

### Global attribute logical properties:

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"><li>• - the hyphen</li><li>• _ the underscore</li><li>• . the period</li></ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <code>xml</code> or any variant (for example <code>XML</code>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>

Property	Type	Meaning
Type	Enumerated type	<p>The Type property constrains the type of data that can be present in the object.</p> <p>There are a limited number of types available directly from the drop down selector. These are;</p> <ul style="list-style-type: none"> <li>• int</li> <li>• string</li> <li>• boolean</li> <li>• hexBinary</li> <li>• dateTime</li> <li>• date</li> <li>• time</li> <li>• decimal</li> <li>• float</li> <li>• (More...)</li> <li>• (New Simple Type)</li> <li>• (New Complex Type)</li> </ul> <p>If you select (More...), this starts the Type Selection wizard. From this wizard you can select any of the available types.</p> <p>If you select (New Simple Type), this starts the New Simple Type wizard which allows you to create an Anonymous simple type which will be based on an existing type. This can be created locally or globally.</p> <p>If you select (New Complex Type), this starts the New Complex Type wizard which allows you to create an Anonymous complex type which can be derived from an existing base type. This can be created locally or globally.</p> <p>For further information about these types, and examples of their use see the XML Schema Part 0: Primer which can be found on the World Wide Web Consortium (W3C) Web site.</p>
Namespace	Enumerated type	<p>Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references.</p> <p>If &lt;no target namespace&gt; is displayed, a namespace has not been set for this object.</p> <p>If the property is inactive, the message set has not been configured to support namespaces.</p> <p>Where the property is active, namespaces that are available for selection will be displayed in the drop down list.</p>

## Value

The *Value* properties are used in conjunction with the *Usage* property in an Attribute Reference or a Local Attribute.

Property	Type	Meaning
Default	Button and String	<p>This property provides the default value for the element or attribute.</p> <p>When writing a fixed-length portion of a message (CWF or fixed-length TDS) if the attribute or element is missing from the message tree, the default value will be inserted into the bit stream in order to preserve the message structure.</p> <p>This property is not used by the message broker when parsing.</p> <p>Note that this behavior differs from the XML Schema specification.</p>
Fixed	Button and String	<p>If a <i>Fixed</i> value is set for an attribute and a message is received, if the attribute exists in the message, the data in the attribute property of the message must match that held in the fixed value of the attribute definition.</p> <p>If the attribute does not exist in the message, the broker will create the attribute and populate it with the data held in the fixed value of the attribute definition.</p>
Interpret Value As	Enumerated Type	<p>Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur.</p> <p>The possible values for this property are:</p> <ul style="list-style-type: none"> <li>• None This is the default value and will indicate that the element or attribute does not have any key value associated with it.</li> <li>• Message Set Identity. Specifies that the value of the element or attribute will correspond to the identifier, name or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendants of the enclosing message are defined. This value will remain in force unless a new element or attribute Message Set Identity field is encountered which resets the Message Set Identity value to this new one.</li> <li>• Message Identity. Specifies that the value of the element or attribute will correspond to the name or alias (in that priority order) that is associated with a message and will act as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity will apply until a new element or attribute Message Identity field is encountered at the same level in the tree. The embedded message may be defined in either the current message set or in a message set identified using a Message Set Identity.</li> </ul> <p><b>Note:</b> This property is only applicable when the type of the object is derived from <code>xsd:string</code>.</p>

#### Global attribute CWF properties:

There are no properties to show.

#### Global attribute XML properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.



Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
Float types - double - float	Integer types - byte - int - long - short - unsignedByte - unsignedInt - unsignedLong - unsignedShort	Interval types - duration <sup>1</sup>	String types - anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token
<b>Note:</b> 1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.			

*XML properties for global attribute and global element binary types:*

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Binary schema types: base64Binary, hexBinary

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>

## Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : <ul style="list-style-type: none"> <li>• CDatahex (the default)</li> <li>• hex</li> <li>• base64</li> </ul>

*XML properties for global attribute and global element boolean types:*

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Boolean schema types: boolean

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>

*XML properties for global attribute and global element dateTime types:*

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>

## Physical representation

Property	Type	Meaning
DateTime Format	String	<p>Specify a format string that specifies the rendering of the value for dateTime elements.</p> <p>The default dateTime format is dependent on the logical type of the object. For information on the defaults for the dateTime format according to the logical type see “DateTime defaults by logical type” on page 709.</p> <p>See “DateTime formats” on page 701 for details of dateTime formats.</p>

*XML properties for global attribute and global element decimal types:*

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element’s identifier. If the element’s identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>

*XML properties for global attribute and global element float types:*

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Float schema types: double, float

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>

*XML properties for global attribute and global element integer types:*

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedLong, unsignedShort

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>

*XML properties for global attribute and global element string types:*

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>

### Global attribute TDS properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
Float types - double - float	Integer types - byte - int - long - short - unsignedByte - unsignedInt - unsignedLong - unsignedShort	Interval types - duration <sup>1</sup>	String types - anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token
<b>Note:</b> 1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.			

*TDS properties for global attribute binary types:*

The TDS Format properties described here apply to:

- Objects: Global Attribute
- Binary schema types: base64Binary, hexBinary

### Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>

### Physical representation

Property	Type	Meaning
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p> <p>Regardless of the value of the <i>Data Element Separator</i> property, either the <i>Length</i> or <i>Length Reference</i> property must be set.</p>

*TDS properties for global attribute boolean types:*

The TDS Format properties described here apply to:

- Objects: Global Attribute
- Boolean schema types: boolean

## Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>

*TDS properties for global attribute dateTime types:*

The TDS Format properties described here apply to:

- Objects: Global Attribute
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>



Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
DateTime Format	String	<p>Specify a template for date and time.</p> <p>The default DateTime format is dependent on the logical type of the object. For information on the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 709.</p> <p>See "DateTime formats" on page 701 for details of date and time formats.</p>

*TDS properties for global attribute decimal types:*

The TDS Format properties described here apply to:

- Objects: Global Attribute
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	<p>Specify a nonzero integer representing the position of an implied decimal point within a number, or specify 0 (zero, the default) to use the formatting of Float or Decimal numbers as specified by the <i>Precision</i> property.</p> <p>If you specify a positive integer, the position of the decimal point is moved left from the right hand side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234</p> <p>If you specify a negative integer, the position of the decimal point is moved right from the right hand side of the number. For example, if you specify -3, the decimal value 1234 represents 1234000</p>

Property	Type	Meaning
Precision	Button and Integer	<p>This value is used if the <i>Virtual Decimal Point</i> property value is 0, indicating that the decimal point is present in the data. It deals with truncation, and specifies the number of digits that should follow the decimal point.</p> <p>Either specify a number of digits:</p> <ul style="list-style-type: none"> <li>• If you set <i>Precision</i> to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123.</li> <li>• If you set <i>Precision</i> to a number less than the number of fractional digits, data is truncated. For example, the value 123.4567 is truncated to 123.45 if you set <i>Precision</i> to 2.</li> <li>• If you set <i>Precision</i> to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value 12.345 is padded to 12.34500 if you set <i>Precision</i> to 5.</li> </ul> <p>Or select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• All Significant Digits (the default): all the significant digits are written to the output bit stream, and there is no decimal separator if there are no fractional digits.</li> <li>• Explicit Decimal Separator: all the significant digits are written to the output bit stream and the decimal separator is always included, even if there are no fractional digits. The decimal separator must be present in the input bitstream, even if no fractional digits are present.</li> </ul>
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to <i>None</i>, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property <i>Leading</i>, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to <i>Trailing</i>, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

*TDS properties for global attribute float types:*

The TDS Format properties described here apply to:

- Objects: Global Attribute
- Float schema types: double, float

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	<p>Specify a nonzero integer representing the position of an implied decimal point within a number, or specify 0 (zero, the default) to use the formatting of Float or Decimal numbers as specified by the <i>Precision</i> property.</p> <p>If you specify a positive integer, the position of the decimal point is moved left from the right hand side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234</p> <p>If you specify a negative integer, the position of the decimal point is moved right from the right hand side of the number. For example, if you specify -3, the decimal value 1234 represents 1234000</p>

Property	Type	Meaning
Precision	Button and Integer	<p>This value is used if the <i>Virtual Decimal Point</i> property value is 0, indicating that the decimal point is present in the data. It deals with truncation, and specifies the number of digits that should follow the decimal point.</p> <p>Either specify a number of digits:</p> <ul style="list-style-type: none"> <li>• If you set <i>Precision</i> to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123.</li> <li>• If you set <i>Precision</i> to a number less than the number of fractional digits, data is truncated. For example, the value 123.4567 is truncated to 123.45 if you set <i>Precision</i> to 2.</li> <li>• If you set <i>Precision</i> to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value 12.345 is padded to 12.34500 if you set <i>Precision</i> to 5.</li> </ul> <p>Or select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• All Significant Digits (the default): all the significant digits are written to the output bit stream, and there is no decimal separator if there are no fractional digits.</li> <li>• Explicit Decimal Separator: all the significant digits are written to the output bit stream and the decimal separator is always included, even if there are no fractional digits. The decimal separator must be present in the input bitstream, even if no fractional digits are present.</li> <li>• Exponential Notation: data is written out to the bit stream as a signed value having the format [sign1]a.bbb[e[sign2]ccc] where: <ul style="list-style-type: none"> <li>– [sign1] is the value of Negative Sign if the value is negative</li> <li>– a is a single decimal digit</li> <li>– bbb is one or more decimal digits</li> <li>– [sign2] is the value of Negative Sign if the exponent is negative</li> <li>– ccc is exactly three decimal digits (the exponent)</li> </ul> </li> </ul> <p>[sign1] and [sign2] are absent if the value and exponent, respectively, are positive.</p> <p>For example, the value -123.456 is represented as -1.23456e002 and the value 0.00012 is represented as 1.2e-004 in the output bit stream (assuming the value of <i>Negative Sign</i> is "-" and <i>Sign Orientation</i> is Leading).</p> <p>The value -0.00012 is represented as 1.2*e*004 if <i>Negative Sign</i> is "*" and <i>Sign Orientation</i> is Trailing.</p>
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	<p>Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.</p>

Property	Type	Meaning
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

*TDS properties for global attribute integer types:*

The TDS Format properties described here apply to:

- Objects: Global Attribute
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedLong, unsignedShort

### Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.

### Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>



Property	Type	Meaning
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

*TDS properties for global attribute string types:*

The TDS Format properties described here apply to:

- Objects: Global Attribute
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.

Property	Type	Meaning
Interpret Element Value	Enumerated Type	<p>Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. This interpretation is generally standard-specific and is therefore hard coded.</p> <p>The possible values for this property are:</p> <ul style="list-style-type: none"> <li>• None (the default value)</li> <li>• EDIFACT Service String</li> <li>• X12 Service String</li> <li>• Message Key</li> <li>• EDIFACT Syntax Level ID</li> <li>• HL7 Service String</li> <li>• HL7 Field Separator</li> </ul> <p><b>Note:</b> The Message Key enumeration has been deprecated</p>

### Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

### Documentation properties for all message set objects:

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### Note:

1. Key, Keyref, and Unique objects do not have documentation properties.

### Global attribute group properties

A global attribute group can have the following properties;

- "Global attribute group logical properties" on page 187
- "Global attribute group CWF properties" on page 206
- "Global attribute group XML properties" on page 213

- “Global attribute group TDS properties” on page 223
- “Documentation properties for all message set objects” on page 175

**Global attribute group logical properties:**

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"> <li>• - the hyphen</li> <li>• _ the underscore</li> <li>• . the period</li> </ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <code>xml</code> or any variant (for example <code>XML</code>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>

**Global attribute group CWF properties:**

There are no properties to show.

**Global attribute group XML properties:**

There are no properties to show.

**Global attribute group TDS properties:**

There are no properties to show.

**Documentation properties for all message set objects:**

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

**Note:**

1. Key, Keyref, and Unique objects do not have documentation properties.

**Global element properties**

A global element can have the following properties;

- “Global element logical properties” on page 187
- “Global element CWF properties” on page 206
- “Global element XML properties” on page 213
- “Global element TDS properties” on page 223

- “Documentation properties for all message set objects” on page 175

**Global element logical properties:**

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"> <li>• - the hyphen</li> <li>• _ the underscore</li> <li>• . the period</li> </ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <i>xml</i> or any variant (for example <i>Xml</i>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>
Type	Enumerated type	<p>The Type property constrains the type of data that can be present in the object.</p> <p>There are a limited number of types available directly from the drop down selector. These are;</p> <ul style="list-style-type: none"> <li>• int</li> <li>• string</li> <li>• boolean</li> <li>• hexBinary</li> <li>• dateTime</li> <li>• date</li> <li>• time</li> <li>• decimal</li> <li>• float</li> <li>• (More...)</li> <li>• (New Simple Type)</li> <li>• (New Complex Type)</li> </ul> <p>If you select (More...), this starts the Type Selection wizard. From this wizard you can select any of the available types.</p> <p>If you select (New Simple Type), this starts the New Simple Type wizard which allows you to create an Anonymous simple type which will be based on an existing type. This can be created locally or globally.</p> <p>If you select (New Complex Type), this starts the New Complex Type wizard which allows you to create an Anonymous complex type which can be derived from an existing base type. This can be created locally or globally.</p> <p>For further information about these types, and examples of their use see the XML Schema Part 0: Primer which can be found on the World Wide Web Consortium (W3C) Web site.</p>

Property	Type	Meaning
Namespace	Enumerated type	<p>Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references.</p> <p>If &lt;no target namespace&gt; is displayed, a namespace has not been set for this object.</p> <p>If the property is inactive, the message set has not been configured to support namespaces.</p> <p>Where the property is active, namespaces that are available for selection will be displayed in the drop down list.</p>

## Value

Property	Type	Meaning
Default	Button and String	<p>This property provides the default value for the element or attribute.</p> <p>When writing a fixed-length portion of a message (CWF or fixed-length TDS) if the attribute or element is missing from the message tree, the default value will be inserted into the bit stream in order to preserve the message structure.</p> <p>This property is not used by the message broker when parsing.</p> <p>Note that this behavior differs from the XML Schema specification.</p>
Fixed	Button and String	<p>If a <i>Fixed</i> value is set for an attribute and a message is received, if the attribute exists in the message, the data in the attribute property of the message must match that held in the fixed value of the attribute definition.</p> <p>If the attribute does not exist in the message, the broker will create the attribute and populate it with the data held in the fixed value of the attribute definition.</p>
Nullable	Check box	Select this if you want the element to be able to be defined as null. This is distinct from being empty where there is no data in the element.
Interpret Value As	Enumerated Type	<p>Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur.</p> <p>The possible values for this property are:</p> <ul style="list-style-type: none"> <li>• None This is the default value and will indicate that the element or attribute does not have any key value associated with it.</li> <li>• Message Set Identity. Specifies that the value of the element or attribute will correspond to the identifier, name or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value will remain in force unless a new element or attribute Message Set Identity field is encountered which resets the Message Set Identity value to this new one.</li> <li>• Message Identity. Specifies that the value of the element or attribute will correspond to the name or alias (in that priority order) that is associated with a message and will act as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity will apply until a new element or attribute Message Identity field is encountered at the same level in the tree. The embedded message may be defined in either the current message set or in a message set identified using a Message Set Identity.</li> </ul> <p><b>Note:</b> This property is only applicable when the type of the object is derived from xsd:string.</p>



## Substitution settings

Substitution Groups provide a means by which one element may be substituted for another in a message. The element which can be substituted is called the 'head' element, and the substitution group is the list of elements that may be used in its place. An element can be in at most one substitution group.

Property	Type	Meaning
Final	Enumerated type	You use this property to limit the set of elements which may belong to its substitution group. <ul style="list-style-type: none"><li>• Empty</li><li>• <i>restriction</i>. Prohibit element substitution by elements whose types are restrictions of the head element's type.</li><li>• <i>extension</i>. Prohibit element substitution by elements whose types are extensions of the head element's type.</li><li>• <i>all</i>. Prohibit substitution by any method.</li></ul>
Block	Enumerated type	You use this property to limit the set of elements which may be substituted for this element in a message.  Select from: <ul style="list-style-type: none"><li>• Empty</li><li>• <i>restriction</i>. Prohibit element substitution by elements whose types are restrictions of the head element's type</li><li>• <i>extension</i>. Prohibit element substitution by elements whose types are extensions of the head element's type</li><li>• <i>substitution</i>. Prohibit element substitution by members of the element's substitution group.</li><li>• <i>#all</i>. Prohibit substitution by any method.</li></ul>
Substitution Group	Enumerated type	Use this property to specify the name of a 'head' element. Setting this property indicates that this element is a member of the substitution group for the 'head' element.
Abstract	Check box	Select this if you do not want the element to appear in the message, but require one of the members of its substitution group to appear in its place.

### Global element CWF properties:

There are no properties to show.

### Global element XML properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
Float types - double - float	Integer types - byte - int - long - short - unsignedByte - unsignedInt - unsignedLong - unsignedShort	Interval types - duration <sup>1</sup>	String types - anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

**Note:**

1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.

*XML properties for global attribute and global element binary types:*

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Binary schema types: base64Binary, hexBinary

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>

## Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : <ul style="list-style-type: none"> <li>• CDatahex (the default)</li> <li>• hex</li> <li>• base64</li> </ul>

*XML properties for global attribute and global element boolean types:*

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Boolean schema types: boolean

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>

*XML properties for global attribute and global element dateTime types:*

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>

## Physical representation

Property	Type	Meaning
DateTime Format	String	<p>Specify a format string that specifies the rendering of the value for dateTime elements.</p> <p>The default dateTime format is dependent on the logical type of the object. For information on the defaults for the dateTime format according to the logical type see “DateTime defaults by logical type” on page 709.</p> <p>See “DateTime formats” on page 701 for details of dateTime formats.</p>

*XML properties for global attribute and global element decimal types:*

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element’s identifier. If the element’s identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>

*XML properties for global attribute and global element float types:*

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Float schema types: double, float

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>

*XML properties for global attribute and global element integer types:*

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedLong, unsignedShort

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>

*XML properties for global attribute and global element string types:*

The XML Wire Format properties described here apply to:

- Objects: Global Attribute, Global Element
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>

### Global element TDS properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
Float types - double - float	Integer types - byte - int - long - short - unsignedByte - unsignedInt - unsignedLong - unsignedShort	Interval types - duration <sup>1</sup>	String types - anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token
<b>Note:</b> 1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.			

*TDS properties for global element binary types:*

The TDS Format properties described here apply to:

- Objects: Global Element
- Binary schema types: base64Binary, hexBinary

### Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>

### Physical representation

Property	Type	Meaning
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p> <p>Regardless of the value of the <i>Data Element Separator</i> property, either the <i>Length</i> or <i>Length Reference</i> property must be set.</p>

*TDS properties for global element boolean types:*

The TDS Format properties described here apply to:

- Objects: Global Element
- Boolean schema types: boolean

## Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>

*TDS properties for global element dateTime types:*

The TDS Format properties described here apply to:

- Objects: Global Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time



## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
DateTime Format	String	<p>Specify a template for date and time.</p> <p>The default DateTime format is dependent on the logical type of the object. For information on the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 709.</p> <p>See "DateTime formats" on page 701 for details of date and time formats.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see “TDS Null handling options” on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a dateTime object to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in “Date Time as string data” on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

*TDS properties for global element decimal types:*

The TDS Format properties described here apply to:

- Objects: Global Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	<p>Specify a nonzero integer representing the position of an implied decimal point within a number, or specify 0 (zero, the default) to use the formatting of Float or Decimal numbers as specified by the <i>Precision</i> property.</p> <p>If you specify a positive integer, the position of the decimal point is moved left from the right hand side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234</p> <p>If you specify a negative integer, the position of the decimal point is moved right from the right hand side of the number. For example, if you specify -3, the decimal value 1234 represents 1234000</p>

Property	Type	Meaning
Precision	Button and Integer	<p>This value is used if the <i>Virtual Decimal Point</i> property value is 0, indicating that the decimal point is present in the data. It deals with truncation, and specifies the number of digits that should follow the decimal point.</p> <p>Either specify a number of digits:</p> <ul style="list-style-type: none"> <li>• If you set <i>Precision</i> to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123.</li> <li>• If you set <i>Precision</i> to a number less than the number of fractional digits, data is truncated. For example, the value 123.4567 is truncated to 123.45 if you set <i>Precision</i> to 2.</li> <li>• If you set <i>Precision</i> to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value 12.345 is padded to 12.34500 if you set <i>Precision</i> to 5.</li> </ul> <p>Or select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• All Significant Digits (the default): all the significant digits are written to the output bit stream, and there is no decimal separator if there are no fractional digits.</li> <li>• Explicit Decimal Separator: all the significant digits are written to the output bit stream and the decimal separator is always included, even if there are no fractional digits. The decimal separator must be present in the input bitstream, even if no fractional digits are present.</li> </ul>
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to <i>None</i>, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property <i>Leading</i>, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to <i>Trailing</i>, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see “TDS Null handling options” on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a dateTime object to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in “Date Time as string data” on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

*TDS properties for global element float types:*

The TDS Format properties described here apply to:

- Objects: Global Element
- Float schema types: double, float

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>



Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	<p>Specify a nonzero integer representing the position of an implied decimal point within a number, or specify 0 (zero, the default) to use the formatting of Float or Decimal numbers as specified by the <i>Precision</i> property.</p> <p>If you specify a positive integer, the position of the decimal point is moved left from the right hand side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234</p> <p>If you specify a negative integer, the position of the decimal point is moved right from the right hand side of the number. For example, if you specify -3, the decimal value 1234 represents 1234000</p>

Property	Type	Meaning
Precision	Button and Integer	<p>This value is used if the <i>Virtual Decimal Point</i> property value is 0, indicating that the decimal point is present in the data. It deals with truncation, and specifies the number of digits that should follow the decimal point.</p> <p>Either specify a number of digits:</p> <ul style="list-style-type: none"> <li>• If you set <i>Precision</i> to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123.</li> <li>• If you set <i>Precision</i> to a number less than the number of fractional digits, data is truncated. For example, the value 123.4567 is truncated to 123.45 if you set <i>Precision</i> to 2.</li> <li>• If you set <i>Precision</i> to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value 12.345 is padded to 12.34500 if you set <i>Precision</i> to 5.</li> </ul> <p>Or select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• All Significant Digits (the default): all the significant digits are written to the output bit stream, and there is no decimal separator if there are no fractional digits.</li> <li>• Explicit Decimal Separator: all the significant digits are written to the output bit stream and the decimal separator is always included, even if there are no fractional digits. The decimal separator must be present in the input bitstream, even if no fractional digits are present.</li> <li>• Exponential Notation: data is written out to the bit stream as a signed value having the format [sign1]a.bbbe[sign2]ccc where: <ul style="list-style-type: none"> <li>– [sign1] is the value of Negative Sign if the value is negative</li> <li>– a is a single decimal digit</li> <li>– bbb is one or more decimal digits</li> <li>– [sign2] is the value of Negative Sign if the exponent is negative</li> <li>– ccc is exactly three decimal digits (the exponent)</li> </ul> </li> </ul> <p>[sign1] and [sign2] are absent if the value and exponent, respectively, are positive.</p> <p>For example, the value -123.456 is represented as -1.23456e002 and the value 0.00012 is represented as 1.2e-004 in the output bit stream (assuming the value of <i>Negative Sign</i> is "-" and <i>Sign Orientation</i> is Leading).</p> <p>The value -0.00012 is represented as 1.2*e*004 if <i>Negative Sign</i> is "*" and <i>Sign Orientation</i> is Trailing.</p>
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	<p>Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.</p>

Property	Type	Meaning
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>• <code>NULLPadFill</code>. This is only valid for fixed length objects. This is the default value.</li> <li>• <code>NULLLogicalValue</code>. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>• <code>NULLLiteralValue</code>. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For <code>dateTime</code> elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see "TDS Null handling options" on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a <code>dateTime</code> object to <code>NULLLogicalValue</code>, you must set this property to an ISO8601 <code>dateTime</code> format. These formats are described in "Date Time as string data" on page 701. For example, specify a value conforming to <code>yyyy-MM-dd'T'HH:mm:ss</code> such as <code>1970-12-01</code>.</p>

*TDS properties for global element integer types:*

The TDS Format properties described here apply to:

- Objects: Global Element
- Integer schema types: `byte`, `int`, `long`, `short`, `unsignedByte`, `unsignedInt`, `unsignedLong`, `unsignedShort`

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is <code>Tagged Delimited</code>, <code>Tagged Fixed Length</code>, or <code>Tagged Encoded Length</code>, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is <code>Tagged Delimited</code>, <code>Tagged Fixed Length</code>, or <code>Tagged Encoded Length</code> then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to <code>Use Data Pattern</code> in the complex type. See "Regular expression syntax" on page 697 for further details.

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>• NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see "TDS Null handling options" on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a dateTime object to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

*TDS properties for global element string types:*

The TDS Format properties described here apply to:

- Objects: Global Element
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>
Interpret Element Value	Enumerated Type	<p>Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. This interpretation is generally standard-specific and is therefore hard coded.</p> <p>The possible values for this property are:</p> <ul style="list-style-type: none"> <li>• None (the default value)</li> <li>• EDIFACT Service String</li> <li>• X12 Service String</li> <li>• Message Key</li> <li>• EDIFACT Syntax Level ID</li> <li>• HL7 Service String</li> <li>• HL7 Field Separator</li> </ul> <p><b>Note:</b> The Message Key enumeration has been deprecated</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>

Property	Type	Meaning
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>



Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>• <code>NULLPadFill</code>. This is only valid for fixed length objects. This is the default value.</li> <li>• <code>NULLLogicalValue</code>. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>• <code>NULLLiteralValue</code>. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For <code>dateTime</code> elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see “TDS Null handling options” on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a <code>dateTime</code> object to <code>NULLLogicalValue</code>, you must set this property to an ISO8601 <code>dateTime</code> format. These formats are described in “Date Time as string data” on page 701. For example, specify a value conforming to <code>yyyy-MM-dd'T'HH:mm:ss</code> such as <code>1970-12-01</code>.</p>

### Documentation properties for all message set objects:

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### Note:

1. Key, Keyref, and Unique objects do not have documentation properties.

### Global group properties

A global element can have the following properties;

- “Global group logical properties” on page 190
- “Global group CWF properties” on page 206
- “Global group XML properties” on page 213
- “Global group TDS properties” on page 224
- “Documentation properties for all message set objects” on page 175

### Global group logical properties:

Valid children in a global group that depend on both **Composition** and **Content Validation** are shown in “Content Validation properties for complex types” on page 182.

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"> <li>• - the hyphen</li> <li>• _ the underscore</li> <li>• . the period</li> </ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <i>xml</i> or any variant (for example <i>Xml</i>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>
Composition	Enumerated type	<p>The following only applies to the element content of a complex type and has no effect on the attribute content of a complex type. Select from:</p> <ul style="list-style-type: none"> <li>• Empty</li> <li>• sequence. If you select this option, you can define children that are elements or groups. These children, if present, must appear in the specified order. They can repeat and can be duplicated.</li> <li>• choice. If you select this option, you can define children that are elements or groups. Only one of the defined children of the complex type can be present, but repeating children are allowed. Use this option if you want to model C unions and COBOL REDEFINES in a Custom Wire Format, or an XML DTD element that uses choice in an XML Wire Format, or a SWIFT field that has more than one option.</li> <li>• all. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can only contain elements - groups are not allowed. An all group can only be used at the top level of a complex type - it cannot be a member of another group within a type.</li> <li>• unorderedSet. If you select this option, you can define only elements as children. The elements can repeat but cannot be duplicated. Child elements can appear in any order.</li> <li>• orderedSet. If you select this option, you can define only elements as children. These elements, if present, must appear in the specified order, and they can repeat but cannot be duplicated. This is the default value for new complex types.</li> <li>• message. If you select this option, you can define only messages as children. They can repeat, but they cannot be duplicated. Like choice, only one of the defined children can be present. If the complex type includes more than one message, the bit stream contains the exact length of the embedded message, and is not padded to the length of the longest. Use this option to model multipart messages, which are used in some industry standards, for example, SWIFT. For more information, see the section on multipart messages in “Multipart messages” on page 28.</li> </ul>

Property	Type	Meaning
Content Validation	Enumerated type	<p><i>Content Validation</i> controls how the broker responds to undeclared content and specifies where the objects that are included within the complex type are defined, if at all. It is used in combination with the <i>Composition</i> property.</p> <p>Options:</p> <ul style="list-style-type: none"> <li>• <i>Closed</i>. The complex type can only contain the child elements that you have added to it.</li> <li>• <i>Open Defined</i>. The complex type can contain any valid element defined within the message set.</li> <li>• <i>Open</i>. The complex type can contain any valid element, not just those that you have added to this complex type.</li> </ul> <p>See “Combinations of Composition and Content Validation” on page 280 for further details of these options.</p>

**Global group CWF properties:**

There are no properties to show.

**Global group XML properties:**

There are no properties to show.

**Global group TDS properties:**

## Field Identification

Property	Type	Meaning
Data Element Separation	Enumerated Type	<p>Specify the method used to separate the data elements within the type. Select one of the following values:</p> <ul style="list-style-type: none"> <li>• Tagged Delimited. This value indicates that all elements within the complex type are identified by a tag, and separated by the value specified in the optional <i>Delimiter</i> property (if specified). You must set the <i>Tag</i> property for all child elements of simple type, and you may set the <i>Delimiter</i> property to a non-empty value. See “Global element TDS properties” on page 223. You must also set the <i>Tag Data Separator</i> or the <i>Length of Tag</i> properties.</li> <li>• Tagged Fixed Length. This value indicates that each element is identified by a tag, and the data has a fixed length. There are no delimiters. You must set the <i>Tag</i> property for each of the child elements of this complex type, and each child element must have a <i>Length</i> or <i>Length Reference</i> property assigned to it. You must also set the <i>Tag Data Separator</i> or the <i>Length of Tag</i> properties.</li> <li>• Tagged Encoded Length. This value indicates that all elements within the complex type are separated by a tag, and a length field follows each tag. There are no delimiters. The tag can be fixed length, as set by <i>Length of Tag</i>, or variable length delimited by the <i>Tag Data Separator</i>. You must also set <i>Length Of Encoded Length</i> so that the parser knows the size of the length field, and set <i>Extra Chars in Encoded Length</i> to tell the parser how many to subtract from the value in <i>Length Of Encoded Length</i> to get the actual length of data that follows the length field.  This method provides a more flexible way of handling ACORD AL3 standard messages than Fixed Length AL3, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard.</li> <li>• All Elements Delimited. This value indicates that all elements within the complex type are separated by a delimiter. You must set the value in the <i>Delimiter</i> property.</li> <li>• Variable Length Elements Delimited. This value indicates that some of the elements within the complex type might be of variable length: if they are, they must be delimited by the value specified in the <i>Delimiter</i> property.</li> <li>• Use Data Pattern. This value indicates that the parser determines the elements by matching the data with the regular expression set in the element or type member <i>Data Pattern</i> property. See “Message definition file properties” on page 174.</li> <li>• Fixed Length. This value indicates that all elements within the complex type are fixed length. The next data element is accessed by adding the value of the <i>Length</i> property to the offset (see “Global element TDS properties” on page 223). If you set the <i>Data Element Separation</i> property of a complex type to Fixed Length, you must also set the <i>Data Element Separation</i> property of all complex children of this type to Fixed Length. Each child element must have a <i>Length</i> or <i>Length Reference</i> property assigned to it.</li> <li>• Fixed Length AL3. This value has a similar meaning to the separation type Fixed Length, but also indicates to the parser that a number of predefined rules with regard to missing optional elements, encoded lengths, and versioning must be applied. If you set the <i>Data Element Separation</i> property of a complex type to Fixed Length AL3, you must also set the <i>Data Element Separation</i> property of all complex children of this type to Fixed Length AL3.</li> <li>• Undefined. This value is set automatically if you set the <i>Type Composition</i> property of a complex type to Message, and you cannot change it to any other value.</li> </ul> <p>Do not set the <i>Type Composition</i> property to Empty, Choice, Unordered Set, Ordered Set, Sequence, or Simple Unordered Set. If you do, you will be unable to check in the type.</p>

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character or string that precedes the data belonging to a group or complex type within the bit stream.
Group Terminator	String	Specify the value of a special character or string that terminates data belonging to a group or a complex type within the bit stream.
Delimiter	String	Specify the value of a special character or string that specifies the delimiter used between data elements.  This property applies only to the delimited <i>Data Element Separation</i> methods (Tagged Delimited, All Elements Delimited, and Variable Elements Delimited).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select if you want delimiters to be suppressed for elements that are missing within a message. Select from: <ul style="list-style-type: none"> <li>• End Of Type. Use this option to suppress the delimiter when an element is missing. For example, if the model has been defined to have up to 3 elements and only 2 are present, the last delimiter can be omitted from the message.</li> <li>• Never. Use this option to ensure that even if optional elements are not present, all delimiters will be written out. This option should be used when the delimiter used to delimit parent and child objects is the same. For example, if an optional child element is missing, message processing applications could not tell where the child elements in a message ended and the next parent element started if the delimiters are all the same.</li> </ul>
Observe Element Length	Check box	Applicable when <i>Data Element Separation</i> is All Elements Delimited and tells the TDS parser to take any <i>Length</i> property of child elements or attributes into account. The default value depends on the setting of the <i>Messaging Standard</i> property (at the message set level) and <i>Data Element Separation</i> properties. <ul style="list-style-type: none"> <li>• Where <i>Data Element Separation</i> is All Elements Delimited and the <i>Messaging Standard</i> is set to TLOG , this property should be set. For all other messaging standards it should not be set.</li> <li>• Where <i>Data Element Separation</i> is Tagged Delimited this property should not be set.</li> <li>• Where <i>Data Element Separation</i> is Tagged Fixed Length, Fixed Length, Fixed Length AL3, or Variable Length Elements Delimited this property will be set and is disabled.</li> <li>• For all other data element separation methods, this property is not set and is disabled.</li> </ul> Any other combination will generate a task list warning.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The <i>Tag Data Separator</i> and <i>Length of Tag</i> properties are mutually exclusive.  If you set the property <i>Tag Data Separator</i> , it overrides <i>Length of Tag</i> .  This property applies only to the tagged <i>Data Element Separation</i> methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).
Length of Tag	Integer	Specify the length of a tag value. When the message is parsed, this allows tags to be extracted from the bit stream if the <i>Tag Data Separator</i> property is not set.  The <i>Tag Data Separator</i> and <i>Length of Tag</i> properties are mutually exclusive. If you set the property <i>Tag Data Separator</i> , it overrides this value.  This property applies only to the tagged <i>Data Element Separation</i> methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).

Property	Type	Meaning
Length of Encoded Length	Integer	<p>Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647.</p> <p>You must set this property if you have set <i>Data Element Separation</i> property to Tagged Encoded Length; it is not valid otherwise.</p> <p>The actual number of data characters parsed also depends on the value of the <i>Extra Chars in Encoded Length</i> property.</p>
Extra Chars in Encoded Length	Integer	<p>(Only valid if the <i>Data Element Separation</i> method is set to Tagged Encoded Length.) Specifies the number of extra characters included in the value found in the length field. (For example, the value in the length might include the size of the length field itself as well as the size of the data field, or it might be the total size of the tag, length, and data fields.)</p> <p>Enter a value from 0 to 2147483647. The parser subtracts this number from the number found in the length field to get the number of data characters that follow the length field.</p> <p>You must set this property if you have set <i>Data Element Separation</i> property to Tagged Encoded Length and the actual number of data characters is less than the value found in the length field.</p>

#### Documentation properties for all message set objects:

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### Note:

1. Key, Keyref, and Unique objects do not have documentation properties.

#### Group reference properties

A group reference can have the following properties;

- “Group reference logical properties” on page 191
- “Group reference CWF properties” on page 206
- “Group reference XML properties” on page 213
- “Group reference TDS properties” on page 227
- “Documentation properties for all message set objects” on page 175

#### Group reference logical properties:

Property	Type	Meaning
Reference Name	Enumerated type	The <i>Reference Name</i> is the name of the object that this object is referring to. The objects available to reference can be selected from the drop down list.

## Occurrence properties

Property	Type	Meaning
Min Occurs	Integer	<p>Specify the minimum number of times that the object can repeat. The default is 1.</p> <p>If the value is set to 0, then the object is optional.</p> <p>With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i>, it must be less than or equal to the value in <i>Max Occurs</i>.</p>
Max Occurs	Integer	<p>Specify the maximum number of times that the object can repeat. The default is 1.</p> <p>If this property is not set, then the object can not occur more than once.</p> <p>If this property is set to 0, it will be interpreted as if the object does not exist in the message.</p> <p>It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.</p>

## Group reference CWF properties: Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.</p> <p>For repeating objects, this property is applied to all instances.</p>



## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

### Group reference XML properties:

There are no properties to show.

### Group reference TDS properties:

#### Field identification

Property	Type	Meaning
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.</p>

## Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>

### Documentation properties for all message set objects:

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

**Note:**

1. Key, Keyref, and Unique objects do not have documentation properties.

## **Key properties**

A key can have the following properties;

- “Key logical properties” on page 192
- “Key CWF properties” on page 207
- “Key XML properties” on page 213
- “Key TDS properties” on page 227
- “Documentation properties for all message set objects” on page 175

### **Key logical properties:**

There are no properties to show.

### **Key CWF properties:**

There are no properties to show.

### **Key XML properties:**

There are no properties to show.

### **Key TDS properties:**

There are no properties to show.

### **Documentation properties for all message set objects:**

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

**Note:**

1. Key, Keyref, and Unique objects do not have documentation properties.

## **Keyref properties**

A keyref can have the following properties;

- “Keyref logical properties” on page 192
- “Keyref CWF properties” on page 207
- “Keyref XML properties” on page 214
- “Keyref TDS properties” on page 228
- “Documentation properties for all message set objects” on page 175

### **Keyref logical properties:**

There are no properties to show.

### **Keyref CWF properties:**

There are no properties to show.

### Keyref XML properties:

There are no properties to show.

### Keyref TDS properties:

There are no properties to show.

### Documentation properties for all message set objects:

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### Note:

1. Key, Keyref, and Unique objects do not have documentation properties.

### Local attribute properties

A local attribute can have the following properties;

- “Local attribute logical properties” on page 192
- “Local attribute CWF properties” on page 207
- “Local attribute XML properties” on page 214
- “Local attribute TDS properties” on page 228
- “Documentation properties for all message set objects” on page 175

#### Local attribute logical properties:

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"><li>• - the hyphen</li><li>• _ the underscore</li><li>• . the period</li></ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <i>xml</i> or any variant (for example <i>Xml</i>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>

Property	Type	Meaning
Type	Enumerated type	<p>The Type property constrains the type of data that can be present in the object.</p> <p>There are a limited number of types available directly from the drop down selector. These are;</p> <ul style="list-style-type: none"> <li>• int</li> <li>• string</li> <li>• boolean</li> <li>• hexBinary</li> <li>• dateTime</li> <li>• date</li> <li>• time</li> <li>• decimal</li> <li>• float</li> <li>• (More...)</li> <li>• (New Simple Type)</li> <li>• (New Complex Type)</li> </ul> <p>If you select (More...), this starts the Type Selection wizard. From this wizard you can select any of the available types.</p> <p>If you select (New Simple Type), this starts the New Simple Type wizard which allows you to create an Anonymous simple type which will be based on an existing type. This can be created locally or globally.</p> <p>If you select (New Complex Type), this starts the New Complex Type wizard which allows you to create an Anonymous complex type which can be derived from an existing base type. This can be created locally or globally.</p> <p>For further information about these types, and examples of their use see the XML Schema Part 0: Primer which can be found on the World Wide Web Consortium (W3C) Web site.</p>
Namespace	Enumerated type	<p>Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references.</p> <p>If &lt;no target namespace&gt; is displayed, a namespace has not been set for this object.</p> <p>If the property is inactive, the message set has not been configured to support namespaces.</p> <p>Where the property is active, namespaces that are available for selection will be displayed in the drop down list.</p>

## Value

The *Value* properties are used in conjunction with the *Usage* property in an Attribute Reference or a Local Attribute.

Property	Type	Meaning
Default	Button and String	<p>This property provides the default value for the element or attribute.</p> <p>When writing a fixed-length portion of a message (CWF or fixed-length TDS) if the attribute or element is missing from the message tree, the default value will be inserted into the bit stream in order to preserve the message structure.</p> <p>This property is not used by the message broker when parsing.</p> <p>Note that this behavior differs from the XML Schema specification.</p>
Fixed	Button and String	<p>If a <i>Fixed</i> value is set for an attribute and a message is received, if the attribute exists in the message, the data in the attribute property of the message must match that held in the fixed value of the attribute definition.</p> <p>If the attribute does not exist in the message, the broker will create the attribute and populate it with the data held in the fixed value of the attribute definition.</p>
Interpret Value As	Enumerated Type	<p>Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur.</p> <p>The possible values for this property are:</p> <ul style="list-style-type: none"> <li>• None This is the default value and will indicate that the element or attribute does not have any key value associated with it.</li> <li>• Message Set Identity. Specifies that the value of the element or attribute will correspond to the identifier, name or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value will remain in force unless a new element or attribute Message Set Identity field is encountered which resets the Message Set Identity value to this new one.</li> <li>• Message Identity. Specifies that the value of the element or attribute will correspond to the name or alias (in that priority order) that is associated with a message and will act as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity will apply until a new element or attribute Message Identity field is encountered at the same level in the tree. The embedded message may be defined in either the current message set or in a message set identified using a Message Set Identity.</li> </ul> <p><b>Note:</b> This property is only applicable when the type of the object is derived from <code>xsd:string</code>.</p>

## Usage properties

Property	Type	Meaning
Usage	Enumerated type	<p>The usage property is used in conjunction with the <i>Value</i> property found in an attribute object. The default for the <i>Usage</i> property is optional.</p> <p>Select from;</p> <ul style="list-style-type: none"> <li>• optional. <ul style="list-style-type: none"> <li>– Where the <i>Value</i> property is set to default and no data has been entered in the <i>Value</i> property, the attribute can appear once and can have any value.</li> <li>– Where the <i>Value</i> property is set to default, the attribute can appear once. If it does not appear, its value will be the data that has been entered in the <i>Value</i> property. If it does appear it will be the value given.</li> <li>– Where the <i>Value</i> property is set to fixed, the attribute can appear once. If it does appear, its value must match the data that has been entered in the <i>Value</i> property. If it does not appear its value will be the data that has been entered in the <i>Value</i> property.</li> </ul> </li> <li>• prohibited. The attribute must not appear.</li> <li>• required. <ul style="list-style-type: none"> <li>– Where the <i>Value</i> property is set to default and no data has been entered in the <i>Value</i> property, the attribute must appear once and can have any value.</li> <li>– Where the <i>Value</i> property is set to fixed, the attribute must appear once and it must match the data that has been entered in the <i>Value</i> property.</li> </ul> </li> </ul>

### Local attribute CWF properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types	Boolean types	DateTime types	Decimal types
<ul style="list-style-type: none"> <li>- base64Binary</li> <li>- hexBinary</li> </ul>	<ul style="list-style-type: none"> <li>- boolean</li> </ul>	<ul style="list-style-type: none"> <li>- date</li> <li>- dateTime</li> <li>- gDay</li> <li>- gMonth</li> <li>- gMonthDay</li> <li>- gYear</li> <li>- gYearMonth</li> <li>- time</li> </ul>	<ul style="list-style-type: none"> <li>- decimal</li> <li>- integer</li> <li>- negativeInteger</li> <li>- nonNegativeInteger</li> <li>- nonPositiveInteger</li> <li>- positiveInteger</li> </ul>

<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>
<p><b>Note:</b></p> <p>1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.</p>			

*CWF properties for attribute reference and local attribute binary types:*

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Binary schema types: base64Binary, hexBinary

### Physical representation

Property	Type	Meaning
Length Count	Button and Integer	<p>If you have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 1.</p> <p>The maximum value that you can specify is 2147483647.</p> <p>The default value is empty (not set).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>

## Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.</p> <p>For repeating objects, this property is applied to all instances.</p>



*CBF properties for attribute reference and local attribute boolean types:*

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Boolean schema types: boolean

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	Specify how the object is aligned from the start of the message. Select one of: <ul style="list-style-type: none"><li>• 1 Bytes. This is the default value.</li><li>• 2 Bytes</li><li>• 4 Bytes</li><li>• 8 Bytes</li><li>• 16 Bytes</li></ul>
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

*CBF properties for attribute reference and local attribute dateTime types:*

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>Fixed Length String. The element's length is determined by other length properties below.</li> <li>Length Encoded String 1. The element's first byte contains the length of the string following the length byte in length units. The maximum length of a Length Encoded String 1 element is 255 length units.</li> <li>Length Encoded String 2. The element's first two bytes contain the length of the string following the 2 length bytes in length units. The maximum length of a Length Encoded String 2 element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding.</li> <li>Null Terminated String. The string ends with the hexadecimal NULL character, X'00'.</li> <li>Packed Decimal. The date<code>Time</code> is coded as a Packed Decimal number. It is valid only if the <i>Date<code>Time</code> Format</i> property represents numeric-only data.</li> <li>Binary. The date<code>Time</code> is encoded as a binary sequence of bytes. If you select this option, the range of symbols that you can specify for the Format String property is less than the range of symbols you can specify if you select a string option (see "Date<code>Time</code> formats" on page 701 for details).</li> <li>Time Seconds. This value supports C <code>time_t</code> and Java Date and Time objects. It is valid only if the <i>Date<code>Time</code> Format</i> property represents numeric-only data.</li> <li>Time Milliseconds. This value supports C <code>time_t</code> and Java Date and Time objects. It is valid only if the <i>Date<code>Time</code> Format</i> property represents numeric-only data.</li> </ul> <p>The default value is fixed length string.</p>
Date <code>Time</code> Format	String	<p>Specify a template for date and time.</p> <p>The default date<code>Time</code> format is dependent on the logical type of the object. For information on the defaults for the date<code>Time</code> format according to the logical type see "Date<code>Time</code> defaults by logical type" on page 709.</p> <p>If you set the <i>Physical Type</i> to Binary, the template is restricted to those components defined in "Date<code>Time</code> as string data" on page 701. If you set the <i>Physical Type</i> to Packed Decimal, Time Seconds, or Time Milliseconds, the template is restricted to those components that represent numbers. In these cases, you must update this <i>Date<code>Time</code> Format</i> property.</p> <p>See "Date<code>Time</code> formats" on page 701 for details of date and time formats.</p>
Length Count	Button and Integer	<p>If you have selected a <i>Physical Type</i> of Fixed Length String, Packed Decimal, or Binary, and have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 1 for all three physical types.</p> <p>The maximum value that you can specify is 256 for Fixed Length String, 10 for Packed Decimal, and 2147483647 for Binary.</p> <p>The default value is 0 (zero).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	<p>If you have set the <i>Physical Type</i> property to Packed Decimal, Time Seconds, or Time Milliseconds, select (the default) or unselect <i>Signed</i>. If you have selected another value for <i>Physical Type</i>, this property is invalid.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to Fixed Length String, select Left Justify (the default value) or Right Justify from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Fixed Length String, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

*CWF properties for attribute reference and local attribute decimal types:*

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	Select one of the following from the drop-down list: <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer and Packed Decimal are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	Enter the number of bytes to specify the element length: <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you set the <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 10.</li> <li>• If you set the <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)</li> </ul>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	Select (the default) or deselect this property. This property is used in conjunction with <i>Sign Orientation</i> .
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to External Decimal and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• <b>Included Leading.</b> This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• <b>Separate Leading.</b> This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• <b>Included Trailing.</b> This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• <b>Separate Trailing.</b> This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
Virtual Decimal Point	Integer	<p>Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point should be moved from its assumed position. For example, a decimal element containing 1234 with a Virtual Decimal value of 3 is 1.234. This is equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>



Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

CWF properties for attribute reference and local attribute float types:

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Float schema types: double, float

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	Select one of the following from the drop-down list: <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Float. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL. This is the default value.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer, Packed Decimal, and Float are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	Enter the number of bytes to specify the element length: <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Float, select a value from the drop-down list. The default value is 8.</li> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you set the <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 10.</li> <li>• If you set the <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)</li> </ul>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	<p>Select or deselect (unsigned, the default) this property. If you have set <i>Physical Type</i> to Float, this is selected. This property is used in conjunction with <i>Sign Orientation</i>.</p>
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to External Decimal and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• <b>Included Leading.</b> This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• <b>Separate Leading.</b> This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• <b>Included Trailing.</b> This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• <b>Separate Trailing.</b> This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
Virtual Decimal Point	Integer	<p>Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point should be moved from its assumed position. For example, a float element containing 1234 with a Virtual Decimal value of 3 is 1.234.</p> <p>This is not applicable if you have set <i>Physical Type</i> to Float.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

CWF properties for attribute reference and local attribute integer types:

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedLong, unsignedShort

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	Select one of the following from the drop-down list: <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer and Packed Decimal are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	Enter the number of bytes to specify the element length: <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you have set <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 6.</li> <li>• If you have set <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 11.</li> </ul>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	Select (the default) or deselect this property. This property is used in conjunction with <i>Sign Orientation</i> .
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to External Decimal and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• <b>Included Leading.</b> This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• <b>Separate Leading.</b> This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• <b>Included Trailing.</b> This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• <b>Separate Trailing.</b> This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>



Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

*CWF properties for attribute reference and local attribute string types:*

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	Select one of the following from the drop-down list: <ul style="list-style-type: none"> <li>• Fixed Length String. The element's length is determined by other length properties below.</li> <li>• Length Encoded String 1. The element's first byte contains the length of the string following the length byte in length units. The maximum length of a Length Encoded String 1 element is 255 length units.</li> <li>• Length Encoded String 2. The element's first two bytes contain the length of the string following the 2 length bytes in length units. The maximum length of a Length Encoded String 2 element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding.</li> <li>• Null Terminated String. The string ends with the hexadecimal NULL character, X'00'.</li> </ul> <p>The default is Fixed Length String.</p>
Length Count	Button and Integer	If you have selected a <i>Physical Type</i> of Fixed Length String or Binary, and have set <i>Length Type</i> to Count, enter the number of length units for the element.  The minimum value that you can specify is 0 (zero), the maximum value that you can specify is 2147483647  The default value is 0 (zero).

Property	Type	Meaning
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message’s CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message’s CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to Fixed Length String, select Left Justify (the default value) or Right Justify from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Fixed Length String, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

### Local attribute XML properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types <ul style="list-style-type: none"> <li>- base64Binary</li> <li>- hexBinary</li> </ul>	Boolean types <ul style="list-style-type: none"> <li>- boolean</li> </ul>	DateTime types <ul style="list-style-type: none"> <li>- date</li> <li>- dateTime</li> <li>- gDay</li> <li>- gMonth</li> <li>- gMonthDay</li> <li>- gYear</li> <li>- gYearMonth</li> <li>- time</li> </ul>	Decimal types <ul style="list-style-type: none"> <li>- decimal</li> <li>- integer</li> <li>- negativeInteger</li> <li>- nonNegativeInteger</li> <li>- nonPositiveInteger</li> <li>- positiveInteger</li> </ul>
Float types <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	Integer types <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	Interval types <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	String types <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>

**Note:**

1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.

*XML properties for attribute reference, element reference, local attribute, local element binary types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Binary schema types: base64Binary, hexBinary

### **Field identification**

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

## Physical representation

Property	Type	Meaning
Encoding	String	<p>Select one of the following values from the drop-down list: :</p> <ul style="list-style-type: none"> <li>• CDatahex (the default)</li> <li>• hex</li> <li>• base64</li> </ul>

*XML properties for attribute reference, element reference, local attribute, local element boolean types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Boolean schema types: boolean

## Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>



Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is <i>id</i>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <i>val</i>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element dateTime types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

### **Field identification**

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

## Physical representation

Property	Type	Meaning
DateTime Format	String	<p>Specify a format string that specifies the rendering of the value for <code>dateTime</code> elements.</p> <p>The default <code>dateTime</code> format is dependent on the logical type of the object. For information on the defaults for the <code>dateTime</code> format according to the logical type see "DateTime defaults by logical type" on page 709.</p> <p>See "DateTime formats" on page 701 for details of <code>dateTime</code> formats.</p>

*XML properties for attribute reference, element reference, local attribute, local element decimal types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Decimal schema types: `decimal`, `integer`, `negativeInteger`, `nonNegativeInteger`, `nonPositiveInteger`, `positiveInteger`

## Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li> <p><b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element.</p> <p>If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</p> </li> <li> <p><b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements.</p> <p>If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</p> </li> <li> <p><b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name</i>, <i>ID Attribute Name</i>, <i>ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li> <p><b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</p> </li> <li> <p><b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name</i>, <i>ID Attribute Name</i>, <i>ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is <i>id</i>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <i>val</i>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element float types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Float schema types: double, float

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li> <p><b>XMLElement.</b> If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element.</p> <p>If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element.</p> <p>This is the default value for element objects.</p> </li> <li> <p><b>XMLAttribute.</b> If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements.</p> <p>If you select this value for more than one object, you must set their <i>XML Name</i> property to different values.</p> <p>This is the default value for attribute objects.</p> </li> <li> <p><b>XMLElementAttrID.</b> If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li> <p><b>XMLElementAttrVal.</b> If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</p> </li> <li> <p><b>XMLElementAttrIDVal.</b> This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element integer types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Integer schema types: `byte`, `int`, `long`, `short`, `unsignedByte`, `unsignedInt`, `unsignedLong`, `unsignedShort`

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>



Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element string types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li> <p><b>XMLElement.</b> If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element.</p> <p>If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element.</p> <p>This is the default value for element objects.</p> </li> <li> <p><b>XMLAttribute.</b> If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements.</p> <p>If you select this value for more than one object, you must set their <i>XML Name</i> property to different values.</p> <p>This is the default value for attribute objects.</p> </li> <li> <p><b>XMLElementAttrID.</b> If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li> <p><b>XMLElementAttrVal.</b> If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</p> </li> <li> <p><b>XMLElementAttrIDVal.</b> This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is <i>id</i>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <i>val</i>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

### Local attribute TDS properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
---	----------------------------	--	---

<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>
--	---	--	--

**Note:**

1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.

*TDS properties for local attribute binary types:*

The TDS Format properties described here apply to:

- Objects: Local Attribute
- Binary schema types: base64Binary, hexBinary

**Field Identification**

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>

## Physical representation

Property	Type	Meaning
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p> <p>Regardless of the value of the <i>Data Element Separator</i> property, either the <i>Length</i> or <i>Length Reference</i> property must be set.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>

*TDS properties for local attribute boolean types:*

The TDS Format properties described here apply to:

- Objects: Local Attribute
- Boolean schema types: boolean

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>

*TDS properties for local attribute dateTime types:*

The TDS Format properties described here apply to:

- Objects: Local Attribute
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
DateTime Format	String	<p>Specify a template for date and time.</p> <p>The default DateTime format is dependent on the logical type of the object. For information on the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 709.</p> <p>See "DateTime formats" on page 701 for details of date and time formats.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

*TDS properties for local attribute decimal types:*

The TDS Format properties described here apply to:

- Objects: Local Attribute



- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>

Property	Type	Meaning
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

## Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	<p>Specify a nonzero integer representing the position of an implied decimal point within a number, or specify 0 (zero, the default) to use the formatting of Float or Decimal numbers as specified by the <i>Precision</i> property.</p> <p>If you specify a positive integer, the position of the decimal point is moved left from the right hand side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234</p> <p>If you specify a negative integer, the position of the decimal point is moved right from the right hand side of the number. For example, if you specify -3, the decimal value 1234 represents 1234000</p>
Precision	Button and Integer	<p>This value is used if the <i>Virtual Decimal Point</i> property value is 0, indicating that the decimal point is present in the data. It deals with truncation, and specifies the number of digits that should follow the decimal point.</p> <p>Either specify a number of digits:</p> <ul style="list-style-type: none"> <li>• If you set <i>Precision</i> to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123.</li> <li>• If you set <i>Precision</i> to a number less than the number of fractional digits, data is truncated. For example, the value 123.4567 is truncated to 123.45 if you set <i>Precision</i> to 2.</li> <li>• If you set <i>Precision</i> to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value 12.345 is padded to 12.34500 if you set <i>Precision</i> to 5.</li> </ul> <p>Or select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• All Significant Digits (the default): all the significant digits are written to the output bit stream, and there is no decimal separator if there are no fractional digits.</li> <li>• Explicit Decimal Separator: all the significant digits are written to the output bit stream and the decimal separator is always included, even if there are no fractional digits. The decimal separator must be present in the input bitstream, even if no fractional digits are present.</li> </ul>
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

*TDS properties for local attribute float types:*

The TDS Format properties described here apply to:

- Objects: Local Attribute
- Float schema types: double, float

### Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>

### Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>

Property	Type	Meaning
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

## Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	<p>Specify a nonzero integer representing the position of an implied decimal point within a number, or specify 0 (zero, the default) to use the formatting of Float or Decimal numbers as specified by the <i>Precision</i> property.</p> <p>If you specify a positive integer, the position of the decimal point is moved left from the right hand side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234</p> <p>If you specify a negative integer, the position of the decimal point is moved right from the right hand side of the number. For example, if you specify -3, the decimal value 1234 represents 1234000</p>
Precision	Button and Integer	<p>This value is used if the <i>Virtual Decimal Point</i> property value is 0, indicating that the decimal point is present in the data. It deals with truncation, and specifies the number of digits that should follow the decimal point.</p> <p>Either specify a number of digits:</p> <ul style="list-style-type: none"> <li>• If you set <i>Precision</i> to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123.</li> <li>• If you set <i>Precision</i> to a number less than the number of fractional digits, data is truncated. For example, the value 123.4567 is truncated to 123.45 if you set <i>Precision</i> to 2.</li> <li>• If you set <i>Precision</i> to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value 12.345 is padded to 12.34500 if you set <i>Precision</i> to 5.</li> </ul> <p>Or select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• All Significant Digits (the default): all the significant digits are written to the output bit stream, and there is no decimal separator if there are no fractional digits.</li> <li>• Explicit Decimal Separator: all the significant digits are written to the output bit stream and the decimal separator is always included, even if there are no fractional digits. The decimal separator must be present in the input bitstream, even if no fractional digits are present.</li> <li>• Exponential Notation: data is written out to the bit stream as a signed value having the format [sign1]a.bbbe[sign2]ccc where: <ul style="list-style-type: none"> <li>– [sign1] is the value of Negative Sign if the value is negative</li> <li>– a is a single decimal digit</li> <li>– bbb is one or more decimal digits</li> <li>– [sign2] is the value of Negative Sign if the exponent is negative</li> <li>– ccc is exactly three decimal digits (the exponent)</li> </ul> </li> </ul> <p>[sign1] and [sign2] are absent if the value and exponent, respectively, are positive.</p> <p>For example, the value -123.456 is represented as -1.23456e002 and the value 0.00012 is represented as 1.2e-004 in the output bit stream (assuming the value of <i>Negative Sign</i> is "-" and <i>Sign Orientation</i> is Leading).</p> <p>The value -0.00012 is represented as 1.2*e*004 if <i>Negative Sign</i> is "*" and <i>Sign Orientation</i> is Trailing.</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

*TDS properties for local attribute integer types:*

The TDS Format properties described here apply to:

- Objects: Local Attribute
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedLong, unsignedShort

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>



Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

## Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

*TDS properties for local attribute string types:*

The TDS Format properties described here apply to:

- Objects: Local Attribute
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.

Property	Type	Meaning
Interpret Element Value	Enumerated Type	<p>Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. This interpretation is generally standard-specific and is therefore hard coded.</p> <p>The possible values for this property are:</p> <ul style="list-style-type: none"> <li>• None (the default value)</li> <li>• EDIFACT Service String</li> <li>• X12 Service String</li> <li>• Message Key</li> <li>• EDIFACT Syntax Level ID</li> <li>• HL7 Service String</li> <li>• HL7 Field Separator</li> </ul> <p><b>Note:</b> The Message Key enumeration has been deprecated</p>

### Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

### Documentation properties for all message set objects:

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### Note:

1. Key, Keyref, and Unique objects do not have documentation properties.

## Local element properties

A local element can have the following properties;

- “Local element logical properties” on page 195
- “Local element CWF properties” on page 208
- “Local element XML properties” on page 214
- “Local element TDS properties” on page 228
- “Documentation properties for all message set objects” on page 175

### Local element logical properties:

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"><li>• - the hyphen</li><li>• _ the underscore</li><li>• . the period</li></ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <code>xml</code> or any variant (for example <code>XML</code>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>

Property	Type	Meaning
Type	Enumerated type	<p>The Type property constrains the type of data that can be present in the object.</p> <p>There are a limited number of types available directly from the drop down selector. These are;</p> <ul style="list-style-type: none"> <li>• int</li> <li>• string</li> <li>• boolean</li> <li>• hexBinary</li> <li>• dateTime</li> <li>• date</li> <li>• time</li> <li>• decimal</li> <li>• float</li> <li>• (More...)</li> <li>• (New Simple Type)</li> <li>• (New Complex Type)</li> </ul> <p>If you select (More...), this starts the Type Selection wizard. From this wizard you can select any of the available types.</p> <p>If you select (New Simple Type), this starts the New Simple Type wizard which allows you to create an Anonymous simple type which will be based on an existing type. This can be created locally or globally.</p> <p>If you select (New Complex Type), this starts the New Complex Type wizard which allows you to create an Anonymous complex type which can be derived from an existing base type. This can be created locally or globally.</p> <p>For further information about these types, and examples of their use see the XML Schema Part 0: Primer which can be found on the World Wide Web Consortium (W3C) Web site.</p>
Namespace	Enumerated type	<p>Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references.</p> <p>If &lt;no target namespace&gt; is displayed, a namespace has not been set for this object.</p> <p>If the property is inactive, the message set has not been configured to support namespaces.</p> <p>Where the property is active, namespaces that are available for selection will be displayed in the drop down list.</p>

## Occurrences

Property	Type	Meaning
Min Occurs	Integer	<p>Specify the minimum number of times that the object can repeat. The default is 1.</p> <p>If the value is set to 0, then the object is optional.</p> <p>With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i>, it must be less than or equal to the value in <i>Max Occurs</i>.</p>

Property	Type	Meaning
Max Occurs	Integer	<p>Specify the maximum number of times that the object can repeat. The default is 1.</p> <p>If this property is not set, then the object can not occur more than once.</p> <p>If this property is set to 0, it will be interpreted as if the object does not exist in the message.</p> <p>It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.</p>

## Value

Property	Type	Meaning
Default	Button and String	<p>This property provides the default value for the element or attribute.</p> <p>When writing a fixed-length portion of a message (CWF or fixed-length TDS) if the attribute or element is missing from the message tree, the default value will be inserted into the bit stream in order to preserve the message structure.</p> <p>This property is not used by the message broker when parsing.</p> <p>Note that this behavior differs from the XML Schema specification.</p>
Fixed	Button and String	<p>If a <i>Fixed</i> value is set for an attribute and a message is received, if the attribute exists in the message, the data in the attribute property of the message must match that held in the fixed value of the attribute definition.</p> <p>If the attribute does not exist in the message, the broker will create the attribute and populate it with the data held in the fixed value of the attribute definition.</p>
Nilable	Check box	<p>Select this if you want the element to be able to be defined as null. This is distinct from being empty where there is no data in the element.</p>
Interpret Value As	Enumerated Type	<p>Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur.</p> <p>The possible values for this property are:</p> <ul style="list-style-type: none"> <li>• None This is the default value and will indicate that the element or attribute does not have any key value associated with it.</li> <li>• Message Set Identity. Specifies that the value of the element or attribute will correspond to the identifier, name or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value will remain in force unless a new element or attribute Message Set Identity field is encountered which resets the Message Set Identity value to this new one.</li> <li>• Message Identity. Specifies that the value of the element or attribute will correspond to the name or alias (in that priority order) that is associated with a message and will act as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity will apply until a new element or attribute Message Identity field is encountered at the same level in the tree. The embedded message may be defined in either the current message set or in a message set identified using a Message Set Identity.</li> </ul> <p><b>Note:</b> This property is only applicable when the type of the object is derived from xsd:string.</p>

## Substitution settings

Substitution Groups provide a means by which one element may be substituted for another in a message. The element which can be substituted is called the 'head' element, and the substitution group is the list of elements that may be used in its place. An element can be in at most one substitution group.

Property	Type	Meaning
Final	Enumerated type	You use this property to limit the set of elements which may belong to its substitution group. <ul style="list-style-type: none"> <li>• Empty</li> <li>• restriction. Prohibit element substitution by elements whose types are restrictions of the head element's type.</li> <li>• extension. Prohibit element substitution by elements whose types are extensions of the head element's type.</li> <li>• all. Prohibit substitution by any method.</li> </ul>
Block	Enumerated type	You use this property to limit the set of elements which may be substituted for this element in a message. <p>Select from:</p> <ul style="list-style-type: none"> <li>• Empty</li> <li>• restriction. Prohibit element substitution by elements whose types are restrictions of the head element's type</li> <li>• extension. Prohibit element substitution by elements whose types are extensions of the head element's type</li> <li>• substitution. Prohibit element substitution by members of the element's substitution group.</li> <li>• #all. Prohibit substitution by any method.</li> </ul>
Substitution Group	Enumerated type	Use this property to specify the name of a 'head' element. Setting this property indicates that this element is a member of the substitution group for the 'head' element.
Abstract	Check box	Select this if you do not want the element to appear in the message, but require one of the members of its substitution group to appear in its place.

### Local element CWF properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types	Boolean types	DateTime types	Decimal types
<ul style="list-style-type: none"> <li>- base64Binary</li> <li>- hexBinary</li> </ul>	<ul style="list-style-type: none"> <li>- boolean</li> </ul>	<ul style="list-style-type: none"> <li>- date</li> <li>- dateTime</li> <li>- gDay</li> <li>- gMonth</li> <li>- gMonthDay</li> <li>- gYear</li> <li>- gYearMonth</li> <li>- time</li> </ul>	<ul style="list-style-type: none"> <li>- decimal</li> <li>- integer</li> <li>- negativeInteger</li> <li>- nonNegativeInteger</li> <li>- nonPositiveInteger</li> <li>- positiveInteger</li> </ul>



<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>
<p><b>Note:</b></p> <p>1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.</p>			

*CWF properties for element reference and local element binary types:*

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Binary schema types: base64Binary, hexBinary

### Physical representation

Property	Type	Meaning
Length Count	Button and Integer	<p>If you have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 1.</p> <p>The maximum value that you can specify is 2147483647.</p> <p>The default value is empty (not set).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>

## Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

*CWF properties for element reference and local element boolean types:*

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Boolean schema types: boolean

## Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

*CWF properties for element reference and local element dateTime types:*

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Fixed Length String. The element's length is determined by other length properties below.</li> <li>• Length Encoded String 1. The element's first byte contains the length of the string following the length byte in length units. The maximum length of a Length Encoded String 1 element is 255 length units.</li> <li>• Length Encoded String 2. The element's first two bytes contain the length of the string following the 2 length bytes in length units. The maximum length of a Length Encoded String 2 element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding.</li> <li>• Null Terminated String. The string ends with the hexadecimal NULL character, X'00'.</li> <li>• Packed Decimal. The dateTime is coded as a Packed Decimal number. It is valid only if the <i>DateTime Format</i> property represents numeric-only data.</li> <li>• Binary. The dateTime is encoded as a binary sequence of bytes. If you select this option, the range of symbols that you can specify for the Format String property is less than the range of symbols you can specify if you select a string option (see "DateTime formats" on page 701 for details).</li> <li>• Time Seconds. This value supports C time_t and Java Date and Time objects. It is valid only if the <i>DateTime Format</i> property represents numeric-only data.</li> <li>• Time Milliseconds. This value supports C time_t and Java Date and Time objects. It is valid only if the <i>DateTime Format</i> property represents numeric-only data.</li> </ul> <p>The default value is fixed length string.</p>

Property	Type	Meaning
DateTime Format	String	<p>Specify a template for date and time.</p> <p>The default dateTime format is dependent on the logical type of the object. For information on the defaults for the dateTime format according to the logical type see “DateTime defaults by logical type” on page 709.</p> <p>If you set the <i>Physical Type</i> to Binary, the template is restricted to those components defined in “DateTime as string data” on page 701. If you set the <i>Physical Type</i> to Packed Decimal, Time Seconds, or Time Milliseconds, the template is restricted to those components that represent numbers. In these cases, you must update this <i>DateTime Format</i> property.</p> <p>See “DateTime formats” on page 701 for details of date and time formats.</p>
Length Count	Button and Integer	<p>If you have selected a <i>Physical Type</i> of Fixed Length String, Packed Decimal, or Binary, and have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 1 for all three physical types.</p> <p>The maximum value that you can specify is 256 for Fixed Length String, 10 for Packed Decimal, and 2147483647 for Binary.</p> <p>The default value is 0 (zero).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message’s CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message’s CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>

Property	Type	Meaning
Signed	Boolean	If you have set the <i>Physical Type</i> property to Packed Decimal, Time Seconds, or Time Milliseconds, select (the default) or unselect <i>Signed</i> . If you have selected another value for <i>Physical Type</i> , this property is invalid.
String Justification	Enumerated Type	If you have set the <i>Physical Type</i> property to Fixed Length String, select Left Justify (the default value) or Right Justify from the drop-down list. If you have selected another value for <i>Physical Type</i> , this property is inactive.
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Fixed Length String, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>• <b>NULLPadFill</b>. This is only valid if <i>Physical Type</i> is Fixed Length String. The field is filled with the value specified by the <i>Padding Character</i>. This is the default value.</li> <li>• <b>NULLLogicalValue</b>. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>• <b>NULLLiteralValue</b>. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. Use this option when the value you have set for <i>Encoding Null Value</i> to specify a null date is not a date<code>Time</code> value, or does not conform to the standard date<code>Time</code> format <code>yyyy-MM-dd 'T'HH:mm:ss</code>.</li> <li>• <b>NULLLiteralFill</b>. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>
Encoding Null Value	String	<p>If you set the <i>Encoding Null</i> property to <b>NULLPadFill</b>, this property is disabled (grayed out).</p> <p>If you set the <i>Encoding Null</i> property to <b>NULLLogicalValue</b>, you must set this property to an ISO8601 date<code>Time</code> format. These formats are described in “Date<code>Time</code> as string data” on page 701. For example, specify a value conforming to <code>yyyy-MM-dd'T'HH:mm:ss</code> such as <code>1970-12-01</code>.</p> <p>If you set the <i>Encoding Null</i> property to <b>NULLLiteralValue</b>, you can enter any value that is the same length as the field.</p> <p>If you set the <i>Encoding Null</i> property to <b>NULLLiteralFill</b>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <b>SPACE</b>, <b>NUL</b>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <i>c</i> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <i>YY</i> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <i>YY</i> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <i>xxxx</i> is a Unicode value specified in hexadecimal format.</li> </ul>

## Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.  Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.  If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.

*CWF properties for element reference and local element decimal types:*

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger



## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>Elements that have <i>Physical Type</i> set to Integer and Packed Decimal are represented in the appropriate WebSphere MQ Encoding value.</li> <li>Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	<p>Enter the number of bytes to specify the element length:</p> <ul style="list-style-type: none"> <li>If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>If you set the <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 10.</li> <li>If you set the <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)</li> </ul>
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>Bytes.</li> <li>Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	<p>Select (the default) or deselect this property. This property is used in conjunction with <i>Sign Orientation</i>.</p>

Property	Type	Meaning
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to External Decimal and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• Included Leading. This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• Separate Leading. This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• Included Trailing. This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• Separate Trailing. This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
Virtual Decimal Point	Integer	<p>Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point should be moved from its assumed position. For example, a decimal element containing 1234 with a Virtual Decimal value of 3 is 1.234. This is equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is External Decimal. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string. You can specify a nonnumeric value for <i>Encoding Null Value</i>.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>

Property	Type	Meaning
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above, with the exception of <code>NULLLiteralFill</code>. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <i>c</i> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <i>YY</i> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <i>YY</i> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <i>xxxx</i> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

*CWF properties for element reference and local element float types:*

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Float schema types: double, float

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Float. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL. This is the default value.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer, Packed Decimal, and Float are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	<p>Enter the number of bytes to specify the element length:</p> <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Float, select a value from the drop-down list. The default value is 8.</li> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you set the <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 10.</li> <li>• If you set the <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)</li> </ul>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	<p>Select or deselect (unsigned, the default) this property. If you have set <i>Physical Type</i> to <i>Float</i>, this is selected. This property is used in conjunction with <i>Sign Orientation</i>.</p>
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to <i>External Decimal</i> and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to <i>Leading</i> or <i>Trailing</i> (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• <b>Included Leading.</b> This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• <b>Separate Leading.</b> This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• <b>Included Trailing.</b> This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• <b>Separate Trailing.</b> This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
Virtual Decimal Point	Integer	<p>Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point should be moved from its assumed position. For example, a float element containing 1234 with a Virtual Decimal value of 3 is 1.234.</p> <p>This is not applicable if you have set <i>Physical Type</i> to Float.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is External Decimal. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string. You can specify a nonnumeric value for <i>Encoding Null Value</i>.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>



Property	Type	Meaning
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above, with the exception of <code>NULLLiteralFill</code>. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <code>c</code> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <code>YY</code> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <code>YY</code> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <code>xxxx</code> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

*CWF properties for element reference and local element integer types:*

The Custom Wire Format properties described here apply to:

- Objects: Element Reference, Local Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedLong, unsignedShort

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer and Packed Decimal are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	<p>Enter the number of bytes to specify the element length:</p> <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you have set <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 6.</li> <li>• If you have set <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 11.</li> </ul>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	Select (the default) or deselect this property. This property is used in conjunction with <i>Sign Orientation</i> .
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to External Decimal and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• <b>Included Leading.</b> This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• <b>Separate Leading.</b> This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• <b>Included Trailing.</b> This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• <b>Separate Trailing.</b> This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is External Decimal. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string. You can specify a nonnumeric value for <i>Encoding Null Value</i>.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>

Property	Type	Meaning
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above, with the exception of <code>NULLLiteralFill</code>. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <i>c</i> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <i>YY</i> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <i>YY</i> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <i>xxxx</i> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

*CWF properties for element reference and local element string types:*

The Custom Wire Format properties described here apply to:

- Objects: Attribute Reference, Local Attribute
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Fixed Length String. The element's length is determined by other length properties below.</li> <li>• Length Encoded String 1. The element's first byte contains the length of the string following the length byte in length units. The maximum length of a Length Encoded String 1 element is 255 length units.</li> <li>• Length Encoded String 2. The element's first two bytes contain the length of the string following the 2 length bytes in length units. The maximum length of a Length Encoded String 2 element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding.</li> <li>• Null Terminated String. The string ends with the hexadecimal NULL character, X'00'.</li> </ul> <p>The default is Fixed Length String.</p>
Length Count	Button and Integer	<p>If you have selected a <i>Physical Type</i> of Fixed Length String or Binary, and have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 0 (zero), the maximum value that you can specify is 2147483647</p> <p>The default value is 0 (zero).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to Fixed Length String, select Left Justify (the default value) or Right Justify from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>



Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Fixed Length String, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is Fixed Length String. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>

Property	Type	Meaning
Encoding Null Value	STRING	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. If specified, its length must be equal to the length of the string element, with the exception of <code>NULLLiteralFill</code>.</p> <p>The default value is empty (not set).</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <code>c</code> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <code>YY</code> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <code>YY</code> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <code>xxxx</code> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

### Local element XML properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
Float types - double - float	Integer types - byte - int - long - short - unsignedByte - unsignedInt - unsignedLong - unsignedShort	Interval types - duration <sup>1</sup>	String types - anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token
<b>Note:</b> 1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.			

*XML properties for attribute reference, element reference, local attribute, local element binary types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Binary schema types: base64Binary, hexBinary

### **Field identification**

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

## Physical representation

Property	Type	Meaning
Encoding	String	<p>Select one of the following values from the drop-down list: :</p> <ul style="list-style-type: none"> <li>• CDatahex (the default)</li> <li>• hex</li> <li>• base64</li> </ul>

*XML properties for attribute reference, element reference, local attribute, local element boolean types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Boolean schema types: boolean

## Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element dateTime types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.



Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

## Physical representation

Property	Type	Meaning
DateTime Format	String	<p>Specify a format string that specifies the rendering of the value for <code>dateTime</code> elements.</p> <p>The default <code>dateTime</code> format is dependent on the logical type of the object. For information on the defaults for the <code>dateTime</code> format according to the logical type see "DateTime defaults by logical type" on page 709.</p> <p>See "DateTime formats" on page 701 for details of <code>dateTime</code> formats.</p>

*XML properties for attribute reference, element reference, local attribute, local element decimal types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Decimal schema types: `decimal`, `integer`, `negativeInteger`, `nonNegativeInteger`, `nonPositiveInteger`, `positiveInteger`

## Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li> <p><b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element.</p> <p>If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</p> </li> <li> <p><b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements.</p> <p>If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</p> </li> <li> <p><b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name</i>, <i>ID Attribute Name</i>, <i>ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li> <p><b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</p> </li> <li> <p><b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name</i>, <i>ID Attribute Name</i>, <i>ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element float types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Float schema types: double, float

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element integer types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- Integer schema types: `byte`, `int`, `long`, `short`, `unsignedByte`, `unsignedInt`, `unsignedLong`, `unsignedShort`

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML properties for attribute reference, element reference, local attribute, local element string types:*

The XML Wire Format properties described here apply to:

- Objects: Attribute Reference, Element Reference, Local Attribute, Local Element
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.



Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is <i>id</i>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <i>val</i>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

### Local element TDS properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - base64Binary - hexBinary	Boolean types - boolean	DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger
---	----------------------------	--	---

<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration<sup>1</sup></li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>
<p><b>Note:</b></p> <p>1. duration: The physical format properties for simple type duration are the same as the physical properties of the String logical types.</p>			

*TDS properties for local element binary types:*

The TDS Format properties described here apply to:

- Objects: Local Attribute, Local Element
- Binary schema types: base64Binary, hexBinary

### Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>

Property	Type	Meaning
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

### Physical representation

Property	Type	Meaning
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p> <p>Regardless of the value of the <i>Data Element Separator</i> property, either the <i>Length</i> or <i>Length Reference</i> property must be set.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>

*TDS properties for local element boolean types:*

The TDS Format properties described here apply to:

- Objects: Local Attribute, Local Element
- Boolean schema types: boolean

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the <i>Data Element Separation</i> of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

*TDS properties for local element dateTime types:*

The TDS Format properties described here apply to:

- Objects: Local Attribute, Local Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the <i>Data Element Separation</i> of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>

Property	Type	Meaning
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
DateTime Format	String	<p>Specify a template for date and time.</p> <p>The default DateTime format is dependent on the logical type of the object. For information on the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 709.</p> <p>See "DateTime formats" on page 701 for details of date and time formats.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>



## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see “TDS Null handling options” on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a dateTime object to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

*TDS properties for local element decimal types:*

The TDS Format properties described here apply to:

- Objects: Local Attribute, Local Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.</p>

Property	Type	Meaning
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

### Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

## Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	<p>Specify a nonzero integer representing the position of an implied decimal point within a number, or specify 0 (zero, the default) to use the formatting of Float or Decimal numbers as specified by the <i>Precision</i> property.</p> <p>If you specify a positive integer, the position of the decimal point is moved left from the right hand side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234</p> <p>If you specify a negative integer, the position of the decimal point is moved right from the right hand side of the number. For example, if you specify -3, the decimal value 1234 represents 1234000</p>
Precision	Button and Integer	<p>This value is used if the <i>Virtual Decimal Point</i> property value is 0, indicating that the decimal point is present in the data. It deals with truncation, and specifies the number of digits that should follow the decimal point.</p> <p>Either specify a number of digits:</p> <ul style="list-style-type: none"> <li>• If you set <i>Precision</i> to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123.</li> <li>• If you set <i>Precision</i> to a number less than the number of fractional digits, data is truncated. For example, the value 123.4567 is truncated to 123.45 if you set <i>Precision</i> to 2.</li> <li>• If you set <i>Precision</i> to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value 12.345 is padded to 12.34500 if you set <i>Precision</i> to 5.</li> </ul> <p>Or select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• All Significant Digits (the default): all the significant digits are written to the output bit stream, and there is no decimal separator if there are no fractional digits.</li> <li>• Explicit Decimal Separator: all the significant digits are written to the output bit stream and the decimal separator is always included, even if there are no fractional digits. The decimal separator must be present in the input bitstream, even if no fractional digits are present.</li> </ul>
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see “TDS Null handling options” on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a dateTime object to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

*TDS properties for local element float types:*

The TDS Format properties described here apply to:

- Objects: Local Attribute, Local Element
- Float schema types: double, float

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.</p>

Property	Type	Meaning
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

### Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

## Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	<p>Specify a nonzero integer representing the position of an implied decimal point within a number, or specify 0 (zero, the default) to use the formatting of Float or Decimal numbers as specified by the <i>Precision</i> property.</p> <p>If you specify a positive integer, the position of the decimal point is moved left from the right hand side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234</p> <p>If you specify a negative integer, the position of the decimal point is moved right from the right hand side of the number. For example, if you specify -3, the decimal value 1234 represents 1234000</p>
Precision	Button and Integer	<p>This value is used if the <i>Virtual Decimal Point</i> property value is 0, indicating that the decimal point is present in the data. It deals with truncation, and specifies the number of digits that should follow the decimal point.</p> <p>Either specify a number of digits:</p> <ul style="list-style-type: none"> <li>• If you set <i>Precision</i> to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123.</li> <li>• If you set <i>Precision</i> to a number less than the number of fractional digits, data is truncated. For example, the value 123.4567 is truncated to 123.45 if you set <i>Precision</i> to 2.</li> <li>• If you set <i>Precision</i> to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value 12.345 is padded to 12.34500 if you set <i>Precision</i> to 5.</li> </ul> <p>Or select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• All Significant Digits (the default): all the significant digits are written to the output bit stream, and there is no decimal separator if there are no fractional digits.</li> <li>• Explicit Decimal Separator: all the significant digits are written to the output bit stream and the decimal separator is always included, even if there are no fractional digits. The decimal separator must be present in the input bitstream, even if no fractional digits are present.</li> <li>• Exponential Notation: data is written out to the bit stream as a signed value having the format [sign1]a.bbbe[sign2]ccc where: <ul style="list-style-type: none"> <li>– [sign1] is the value of Negative Sign if the value is negative</li> <li>– a is a single decimal digit</li> <li>– bbb is one or more decimal digits</li> <li>– [sign2] is the value of Negative Sign if the exponent is negative</li> <li>– ccc is exactly three decimal digits (the exponent)</li> </ul> </li> </ul> <p>[sign1] and [sign2] are absent if the value and exponent, respectively, are positive.</p> <p>For example, the value -123.456 is represented as -1.23456e002 and the value 0.00012 is represented as 1.2e-004 in the output bit stream (assuming the value of <i>Negative Sign</i> is "-" and <i>Sign Orientation</i> is Leading).</p> <p>The value -0.00012 is represented as 1.2*e*004 if <i>Negative Sign</i> is "*" and <i>Sign Orientation</i> is Trailing.</p>



Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

### Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>• <code>NULLPadFill</code>. This is only valid for fixed length objects. This is the default value.</li> <li>• <code>NULLLogicalValue</code>. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>• <code>NULLLiteralValue</code>. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For date<code>Time</code> elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see "TDS Null handling options" on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a date<code>Time</code> object to <code>NULLLogicalValue</code>, you must set this property to an ISO8601 date<code>Time</code> format. These formats are described in "Date<code>Time</code> as string data" on page 701. For example, specify a value conforming to <code>yyyy-MM-dd'T'HH:mm:ss</code> such as 1970-12-01.</p>

*TDS properties for local element integer types:*

The TDS Format properties described here apply to:

- Objects: Local Attribute, Local Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedLong, unsignedShort

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the <i>Data Element Separation</i> of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>

Property	Type	Meaning
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

## Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>• <code>NULLPadFill</code>. This is only valid for fixed length objects. This is the default value.</li> <li>• <code>NULLLogicalValue</code>. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>• <code>NULLLiteralValue</code>. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For <code>dateTime</code> elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see "TDS Null handling options" on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a <code>dateTime</code> object to <code>NULLLogicalValue</code>, you must set this property to an ISO8601 <code>dateTime</code> format. These formats are described in "Date/Time as string data" on page 701. For example, specify a value conforming to <code>yyyy-MM-dd'T'HH:mm:ss</code> such as <code>1970-12-01</code>.</p>

*TDS properties for local element string types:*

The TDS Format properties described here apply to:

- Objects: Local Attribute, Local Element
- String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

## Field Identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>
Interpret Element Value	Enumerated Type	<p>Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. This interpretation is generally standard-specific and is therefore hard coded.</p> <p>The possible values for this property are:</p> <ul style="list-style-type: none"> <li>• None (the default value)</li> <li>• EDIFACT Service String</li> <li>• X12 Service String</li> <li>• Message Key</li> <li>• EDIFACT Syntax Level ID</li> <li>• HL7 Service String</li> <li>• HL7 Field Separator</li> </ul> <p><b>Note:</b> The Message Key enumeration has been deprecated</p>
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the <i>Data Element Separation</i> of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>



## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see “TDS Null handling options” on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a dateTime object to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

### Documentation properties for all message set objects:

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### Note:

1. Key, Keyref, and Unique objects do not have documentation properties.

### Local group properties

A local group can have the following properties;

- “Local group logical properties” on page 198
- “Local group CWF properties” on page 209
- “Local group XML properties” on page 215
- “Local group TDS properties” on page 229
- “Documentation properties for all message set objects” on page 175

#### Local group logical properties:

Valid children in a local group that depend on both **Composition** and **Content Validation** are shown in “Content Validation properties for complex types” on page 182.

Property	Type	Meaning
Composition	Enumerated type	<p>The following only applies to the element content of a complex type and has no effect on the attribute content of a complex type. Select from:</p> <ul style="list-style-type: none"> <li>• Empty</li> <li>• sequence. If you select this option, you can define children that are elements or groups. These children, if present, must appear in the specified order. They can repeat and can be duplicated.</li> <li>• choice. If you select this option, you can define children that are elements or groups. Only one of the defined children of the complex type can be present, but repeating children are allowed. Use this option if you want to model C unions and COBOL REDEFINES in a Custom Wire Format, or an XML DTD element that uses choice in an XML Wire Format, or a SWIFT field that has more than one option.</li> <li>• all. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can only contain elements - groups are not allowed. An all group can only be used at the top level of a complex type - it cannot be a member of another group within a type.</li> <li>• unorderedSet. If you select this option, you can define only elements as children. The elements can repeat but cannot be duplicated. Child elements can appear in any order.</li> <li>• orderedSet. If you select this option, you can define only elements as children. These elements, if present, must appear in the specified order, and they can repeat but cannot be duplicated. This is the default value for new complex types.</li> <li>• message. If you select this option, you can define only messages as children. They can repeat, but they cannot be duplicated. Like choice, only one of the defined children can be present. If the complex type includes more than one message, the bit stream contains the exact length of the embedded message, and is not padded to the length of the longest. Use this option to model multipart messages, which are used in some industry standards, for example, SWIFT. For more information, see the section on multipart messages in “Multipart messages” on page 28.</li> </ul>
Content Validation	Enumerated type	<p><i>Content Validation</i> controls how the broker responds to undeclared content and specifies where the objects that are included within the complex type are defined, if at all. It is used in combination with the <i>Composition</i> property.</p> <p>Options:</p> <ul style="list-style-type: none"> <li>• Closed. The complex type can only contain the child elements that you have added to it.</li> <li>• Open Defined. The complex type can contain any valid element defined within the message set.</li> <li>• Open. The complex type can contain any valid element, not just those that you have added to this complex type.</li> </ul> <p>See “Combinations of Composition and Content Validation” on page 280 for further details of these options.</p>

## Occurrences

Property	Type	Meaning
Min Occurs	Integer	<p>Specify the minimum number of times that the object can repeat. The default is 1.</p> <p>If the value is set to 0, then the object is optional.</p> <p>With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i>, it must be less than or equal to the value in <i>Max Occurs</i>.</p>
Max Occurs	Integer	<p>Specify the maximum number of times that the object can repeat. The default is 1.</p> <p>If this property is not set, then the object can not occur more than once.</p> <p>If this property is set to 0, it will be interpreted as if the object does not exist in the message.</p> <p>It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.</p>

### Local group CWF properties: Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

### Local group XML properties:

There are no properties to show.

### Local group TDS properties:

## Field Identification

Property	Type	Meaning
Data Element Separation	Enumerated Type	<p>Specify the method used to separate the data elements within the type. Select one of the following values:</p> <ul style="list-style-type: none"> <li>• Tagged Delimited. This value indicates that all elements within the complex type are identified by a tag, and separated by the value specified in the optional <i>Delimiter</i> property (if specified). You must set the <i>Tag</i> property for all child elements of simple type, and you may set the <i>Delimiter</i> property to a non-empty value. See “Global element TDS properties” on page 223. You must also set the <i>Tag Data Separator</i> or the <i>Length of Tag</i> properties.</li> <li>• Tagged Fixed Length. This value indicates that each element is identified by a tag, and the data has a fixed length. There are no delimiters. You must set the <i>Tag</i> property for each of the child elements of this complex type, and each child element must have a <i>Length</i> or <i>Length Reference</i> property assigned to it. You must also set the <i>Tag Data Separator</i> or the <i>Length of Tag</i> properties.</li> <li>• Tagged Encoded Length. This value indicates that all elements within the complex type are separated by a tag, and a length field follows each tag. There are no delimiters. The tag can be fixed length, as set by <i>Length of Tag</i>, or variable length delimited by the <i>Tag Data Separator</i>. You must also set <i>Length Of Encoded Length</i> so that the parser knows the size of the length field, and set <i>Extra Chars in Encoded Length</i> to tell the parser how many to subtract from the value in <i>Length Of Encoded Length</i> to get the actual length of data that follows the length field.  This method provides a more flexible way of handling ACORD AL3 standard messages than Fixed Length AL3, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard.</li> <li>• All Elements Delimited. This value indicates that all elements within the complex type are separated by a delimiter. You must set the value in the <i>Delimiter</i> property.</li> <li>• Variable Length Elements Delimited. This value indicates that some of the elements within the complex type might be of variable length: if they are, they must be delimited by the value specified in the <i>Delimiter</i> property.</li> <li>• Use Data Pattern. This value indicates that the parser determines the elements by matching the data with the regular expression set in the element or type member <i>Data Pattern</i> property. See “Message definition file properties” on page 174.</li> <li>• Fixed Length. This value indicates that all elements within the complex type are fixed length. The next data element is accessed by adding the value of the <i>Length</i> property to the offset (see “Global element TDS properties” on page 223). If you set the <i>Data Element Separation</i> property of a complex type to Fixed Length, you must also set the <i>Data Element Separation</i> property of all complex children of this type to Fixed Length. Each child element must have a <i>Length</i> or <i>Length Reference</i> property assigned to it.</li> <li>• Fixed Length AL3. This value has a similar meaning to the separation type Fixed Length, but also indicates to the parser that a number of predefined rules with regard to missing optional elements, encoded lengths, and versioning must be applied. If you set the <i>Data Element Separation</i> property of a complex type to Fixed Length AL3, you must also set the <i>Data Element Separation</i> property of all complex children of this type to Fixed Length AL3.</li> <li>• Undefined. This value is set automatically if you set the <i>Type Composition</i> property of a complex type to Message, and you cannot change it to any other value.</li> </ul> <p>Do not set the <i>Type Composition</i> property to Empty, Choice, Unordered Set, Ordered Set, Sequence, or Simple Unordered Set. If you do, you will be unable to check in the type.</p>

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character or string that precedes the data belonging to a group or complex type within the bit stream.
Group Terminator	String	Specify the value of a special character or string that terminates data belonging to a group or a complex type within the bit stream.
Delimiter	String	Specify the value of a special character or string that specifies the delimiter used between data elements.  This property applies only to the delimited <i>Data Element Separation</i> methods (Tagged Delimited, All Elements Delimited, and Variable Elements Delimited).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select if you want delimiters to be suppressed for elements that are missing within a message. Select from: <ul style="list-style-type: none"> <li>• End Of Type. Use this option to suppress the delimiter when an element is missing. For example, if the model has been defined to have up to 3 elements and only 2 are present, the last delimiter can be omitted from the message.</li> <li>• Never. Use this option to ensure that even if optional elements are not present, all delimiters will be written out. This option should be used when the delimiter used to delimit parent and child objects is the same. For example, if an optional child element is missing, message processing applications could not tell where the child elements in a message ended and the next parent element started if the delimiters are all the same.</li> </ul>
Observe Element Length	Check box	Applicable when <i>Data Element Separation</i> is All Elements Delimited and tells the TDS parser to take any <i>Length</i> property of child elements or attributes into account. The default value depends on the setting of the <i>Messaging Standard</i> property (at the message set level) and <i>Data Element Separation</i> properties. <ul style="list-style-type: none"> <li>• Where <i>Data Element Separation</i> is All Elements Delimited and the <i>Messaging Standard</i> is set to TLOG , this property should be set. For all other messaging standards it should not be set.</li> <li>• Where <i>Data Element Separation</i> is Tagged Delimited this property should not be set.</li> <li>• Where <i>Data Element Separation</i> is Tagged Fixed Length, Fixed Length, Fixed Length AL3, or Variable Length Elements Delimited this property will be set and is disabled.</li> <li>• For all other data element separation methods, this property is not set and is disabled.</li> </ul> Any other combination will generate a task list warning.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The <i>Tag Data Separator</i> and <i>Length of Tag</i> properties are mutually exclusive.  If you set the property <i>Tag Data Separator</i> , it overrides <i>Length of Tag</i> .  This property applies only to the tagged <i>Data Element Separation</i> methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).
Length of Tag	Integer	Specify the length of a tag value. When the message is parsed, this allows tags to be extracted from the bit stream if the <i>Tag Data Separator</i> property is not set.  The <i>Tag Data Separator</i> and <i>Length of Tag</i> properties are mutually exclusive. If you set the property <i>Tag Data Separator</i> , it overrides this value.  This property applies only to the tagged <i>Data Element Separation</i> methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).

Property	Type	Meaning
Length of Encoded Length	Integer	<p>Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647.</p> <p>You must set this property if you have set <i>Data Element Separation</i> property to Tagged Encoded Length; it is not valid otherwise.</p> <p>The actual number of data characters parsed also depends on the value of the <i>Extra Chars in Encoded Length</i> property.</p>
Extra Chars in Encoded Length	Integer	<p>(Only valid if the <i>Data Element Separation</i> method is set to Tagged Encoded Length.) Specifies the number of extra characters included in the value found in the length field. (For example, the value in the length might include the size of the length field itself as well as the size of the data field, or it might be the total size of the tag, length, and data fields.)</p> <p>Enter a value from 0 to 2147483647. The parser subtracts this number from the number found in the length field to get the number of data characters that follow the length field.</p> <p>You must set this property if you have set <i>Data Element Separation</i> property to Tagged Encoded Length and the actual number of data characters is less than the value found in the length field.</p>
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.</p>

### Documentation properties for all message set objects:

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### Note:

1. Key, Keyref, and Unique objects do not have documentation properties.

## Message properties

A message can have the following properties;

- “Message logical properties” on page 200
- “Message CWF properties” on page 209
- “Message XML properties” on page 215
- “Message TDS properties” on page 232
- “Documentation properties for all message set objects” on page 175

### Message logical properties:

This section describes the logical properties of a message.

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"><li>• - the hyphen</li><li>• _ the underscore</li><li>• . the period</li></ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <code>xml</code> or any variant (for example <code>XML</code>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>
Message Alias	String	<p>Specify an alternative unique value that identifies the message. This property is only required if you are using the Message Identity technique to identify embedded messages, and the bit stream does not contain the actual message name.</p>

### Message CWF properties:

There are no properties to show.

### Message XML properties: Namespace schema locations

This property is only active if namespaces have been enabled.

Property	Type	Meaning
Namespace URI	String	<p>A unique string, usually in the form of a URL that identifies the schema for this</p> <p>If namespaces have not been enabled, this property will display <code>&lt;no target namespace&gt;</code>.</p> <p>This property will override the same property at the message set level.</p>



Property	Type	Meaning
Schema location	String	Enter the location of the schema for the associated namespace name that will be used to validate objects within the namespace.

## XML declarations

Property	Type	Meaning
Output Namespace Declaration	Enumerated Type	<p>The <i>Output Namespace Declaration</i> property controls where the namespace declarations will be placed in the output XML document.</p> <p>Select from:</p> <ul style="list-style-type: none"> <li>At start of document. Declarations for all of the entries in the <i>Namespace schema locations</i> table above will be output as attributes of the message in the output XML document. The disadvantage of this option is that in some cases unnecessary declarations may be output.</li> <li>As required. Declarations will only be output when required by an element or attribute that is in that namespace. The disadvantage of this option is that the same namespace declaration may need to be output more than once in the output XML document.</li> </ul> <p>The default option is At start of document.</p> <p>This property is only active if namespaces are enabled for this message set.</p>

## XML document type settings

Property	Type	Meaning
DOCTYPE System ID	String	<p>Specify the System ID for DOCTYPE external DTD subset. It overrides the equivalent message set property setting for this particular message.</p> <p>If the message set property <i>Suppress DOCTYPE</i> is set to Yes, this parameter is ignored and cannot be changed (the field is disabled) .</p> <p>The default value is the value that you specified for the <i>DOCTYPE System ID</i> property for the message set.</p>
DOCTYPE Public ID	String	<p>Specify the Public ID for DOCTYPE external DTD subset. It overrides the equivalent message set property setting for this particular message.</p> <p>If the message set property <i>Suppress DOCTYPE</i> is set to Yes, this parameter is ignored and cannot be changed (the field is disabled) .</p> <p>The default value is the value that you specified for the <i>DOCTYPE Public ID</i> property for the message set.</p>
DOCTYPE Text	String	<p>Enter optional additional text to include within the DOCTYPE. It overrides the message set property for this particular message.</p> <p>If the message set property <i>Suppress DOCTYPE</i> is set to Yes, this parameter is ignored and cannot be changed (the field is disabled) .</p> <p>For more information, see “In-line DTDs and the DOCTYPE text property” on page 162.</p> <p>The default value is the value that you specified for the <i>DOCTYPE Text</i> property for the message set.</p>

Property	Type	Meaning
Root Tag Name	String	Specify the name of the root tag for a message bit stream XML document. It overrides the message set property set for this message.  The default value is the value that you specified for the <i>Root Tag Name</i> property for the message set.

## Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list: <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element.  If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>.  If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name</i>, <i>ID Attribute Name</i>, <i>ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>
XML Name	String	Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).  This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.  You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.  If you do not set a value, it defaults to that of the element’s identifier. If the element’s identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).

Property	Type	Meaning
ID Attribute Name	String	Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i> , <i>XMLAttribute</i> , or <i>XMLElementAttrVal</i> .  The default value is <i>id</i> .
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i> , <i>XMLAttribute</i> , or <i>XMLElementAttrVal</i> .  The default value is the identifier of the child.

#### Message TDS properties:

Property	Type	Meaning
Message Key	String	Specify an alternative unique value that identifies the message in the bit stream. This property is required if the message is embedded within another message. <b>Note:</b> In Version 6.0 the use of Message Key has been deprecated for identifying an embedded message. You now have the option of identifying an embedded message using a Message Identity.

#### Documentation properties for all message set objects:

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### Note:

1. Key, Keyref, and Unique objects do not have documentation properties.

#### Simple type properties

A simple type can have the following properties;

- “Simple type logical properties” on page 201
- “Simple type CWF properties” on page 209
- “Simple type XML properties” on page 218
- “Simple type TDS properties” on page 232
- “Documentation properties for all message set objects” on page 175

A simple type can also have “Simple type logical value constraints” on page 201.

#### Simple type logical properties:

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"> <li>• - the hyphen</li> <li>• _ the underscore</li> <li>• . the period</li> </ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <i>xml</i> or any variant (for example <i>Xml</i>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>
Base Type	Enumerated type	<p>This property only applies to a simple type restriction.</p> <p>You can use this property to select a base type that is used as the starting point to define a new simple type that is derived by setting additional value constraints.</p>
Item Type	Enumerated type	<p>This property only applies to a simple type list.</p> <p>You can use this property to select the type that is used as the item type of the list.</p>
Variety	Enumerated type	<p>This property displays the variety of the simple type you have selected, either atomic, list, or union.</p>

A simple type can also have “Simple type logical value constraints” on page 201.

*Simple type logical value constraints:*

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types	Boolean types	DateTime types	Decimal types
<ul style="list-style-type: none"> <li>- base64Binary</li> <li>- hexBinary</li> </ul>	<ul style="list-style-type: none"> <li>- boolean</li> </ul>	<ul style="list-style-type: none"> <li>- date</li> <li>- dateTime</li> <li>- gDay</li> <li>- gMonth</li> <li>- gMonthDay</li> <li>- gYear</li> <li>- gYearMonth</li> <li>- time</li> </ul>	<ul style="list-style-type: none"> <li>- decimal</li> <li>- integer</li> <li>- negativeInteger</li> <li>- nonNegativeInteger</li> <li>- nonPositiveInteger</li> <li>- positiveInteger</li> </ul>

<p>Float types</p> <ul style="list-style-type: none"> <li>- double</li> <li>- float</li> </ul>	<p>Integer types</p> <ul style="list-style-type: none"> <li>- byte</li> <li>- int</li> <li>- long</li> <li>- short</li> <li>- unsignedByte</li> <li>- unsignedInt</li> <li>- unsignedLong</li> <li>- unsignedShort</li> </ul>	<p>Interval types</p> <ul style="list-style-type: none"> <li>- duration</li> </ul>	<p>String types</p> <ul style="list-style-type: none"> <li>- anyURI</li> <li>- ENTITIES</li> <li>- ENTITY</li> <li>- ID</li> <li>- IDREF</li> <li>- IDREFS</li> <li>- language</li> <li>- Name</li> <li>- NCName</li> <li>- NMTOKEN</li> <li>- NMTOKENS</li> <li>- normalizedString</li> <li>- NOTATION</li> <li>- QName</li> <li>- string</li> <li>- token</li> </ul>
--	---	--	--

*Logical properties for value constraints for simple type binary types:*

The simple type value constraint properties described here apply to:

- Objects: Simple types
- Binary schema types: base64Binary, hexBinary

### Length constraints

Property	Type	Meaning
Length	Integer	<p>The length property is used to specify the exact length of the simple type in bytes or characters.</p> <p>The value must be greater than 0 and less than 2147483648.</p>
Min	Integer	<p>The Min property is used to specify the minimum length of the simple type in bytes or characters.</p> <p>The value must be greater than 0 and less than 2147483648.</p>
Max	Integer	<p>The Max property is used to specify the maximum length of the simple type in bytes or characters.</p> <p>The value must be greater than 0 and less than 2147483648.</p>

Property	Type	Meaning
White Space	Enumerated type	<p>The <i>White Space</i> controls the processing of white space characters received for this type.</p> <p>The value of <i>White Space</i> must be one of;</p> <ul style="list-style-type: none"> <li>• preserve. If you set the property to preserve, all white space characters including carriage return, line feed and tab are preserved.</li> <li>• replace. If you set the property to replace, all carriage return, line feed and tab characters are replaced with a space character.</li> <li>• collapse. If you set the property to collapse, all carriage return, line feed and tab characters are replaced with a space character. Any adjacent white space characters are then collapsed to a single space character and any leading or trailing spaces are stripped from the data.</li> </ul>

## Enumerations

Property	Type	Meaning
Enumerations	String	<p><i>Enumerations</i> constrain the values to the list that is specified in this property. For example, you could create a simple type called <i>RainbowColors</i>. You would then add Red, Orange, Yellow, Green, Blue, Indigo, and Violet to the enumerations list.</p> <p>You will need to ensure that you have all variations of the data that you are likely to receive in the message defined in the list. For example, Yellow, yellow, yel, y could be possible variations of a single color.</p> <p>Select <b>Add</b> to add a default enumeration. Overtyping the default with the data you require.</p> <p>If you need to change an entry, select the entry, then click on the entry a second time (as distinct from double-click). The selected entry can then be updated.</p>

## Patterns

Property	Type	Meaning
Patterns	String	<p><i>Patterns</i> are a regular expression or a series of regular expressions used to constrain the data within the simple type. For further information on patterns and their syntax see “Using regular expressions to parse data elements” on page 695.</p> <p>Select <b>Add</b> to add a default pattern. Overtyping the default with the data you require.</p> <p>If you need to change an entry, select the entry, then click on the entry a second time (as distinct from double-click). The selected entry can then be updated.</p>

*Logical properties for value constraints for simple type boolean types:*

The simple type value constraint properties described here apply to:

- Objects: Simple types
- Boolean schema types: boolean

Property	Type	Meaning
White Space	Enumerated type	<p>The <i>White Space</i> controls the processing of white space characters received for this type.</p> <p>The value of <i>White Space</i> must be one of;</p> <ul style="list-style-type: none"> <li>• preserve. If you set the property to preserve, all white space characters including carriage return, line feed and tab are preserved.</li> <li>• replace. If you set the property to replace, all carriage return, line feed and tab characters are replaced with a space character.</li> <li>• collapse. If you set the property to collapse, all carriage return, line feed and tab characters are replaced with a space character. Any adjacent white space characters are then collapsed to a single space character and any leading or trailing spaces are stripped from the data.</li> </ul>

## Patterns

Property	Type	Meaning
Patterns	String	<p><i>Patterns</i> are a regular expression or a series of regular expressions used to constrain the data within the simple type. For further information on patterns and their syntax see “Using regular expressions to parse data elements” on page 695.</p> <p>Select <b>Add</b> to add a default pattern. Overtyping the default with the data you require.</p> <p>If you need to change an entry, select the entry, then click on the entry a second time (as distinct from double-click). The selected entry can then be updated.</p>

*Logical properties for value constraints for simple type dateTime types:*

The simple type value constraint properties described here apply to:

- Objects: Simple types
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

## Inclusive Constraints

Property	Type	Meaning
Min	Integer	<p>The <i>Min</i> property is used to specify the minimum value for which the data in the message must be greater than or equal to.</p> <p>When this value is set it cannot be equal to or greater than the <i>Max</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Min</i> Inclusive Constraint and <i>Min</i> Exclusive Constraint properties together for the same simple type.</p>
Max	Integer	<p>The <i>Max</i> property is used to specify the maximum value for which the data in the message must be less than or equal to.</p> <p>When this value is set it cannot be equal to or less than the <i>Min</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Max</i> Inclusive Constraint and <i>Max</i> Exclusive Constraint properties together for the same simple type.</p>

## Exclusive Constraints

Property	Type	Meaning
Min	Integer	<p>The <i>Min</i> property is used to specify the minimum value for which the data in the message must be greater than.</p> <p>When this value is set it cannot be equal to or greater than the <i>Max</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Min</i> Inclusive Constraint and <i>Min</i> Exclusive Constraint properties together for the same simple type.</p>

Property	Type	Meaning
Max	Integer	<p>The <i>Max</i> property is used to specify the maximum value for which the data in the message must be less than.</p> <p>When this value is set it cannot be equal to or less than the <i>Min Inclusive Constraint</i> property.</p> <p>You cannot specify both <i>Max Inclusive Constraint</i> and <i>Max Exclusive Constraint</i> properties together for the same simple type.</p>

Property	Type	Meaning
White Space	Enumerated type	<p>The <i>White Space</i> controls the processing of white space characters received for this type.</p> <p>The value of <i>White Space</i> must be one of;</p> <ul style="list-style-type: none"> <li>• preserve. If you set the property to preserve, all white space characters including carriage return, line feed and tab are preserved.</li> <li>• replace. If you set the property to replace, all carriage return, line feed and tab characters are replaced with a space character.</li> <li>• collapse. If you set the property to collapse, all carriage return, line feed and tab characters are replaced with a space character. Any adjacent white space characters are then collapsed to a single space character and any leading or trailing spaces are stripped from the data.</li> </ul>

## Enumerations

Property	Type	Meaning
Enumerations	String	<p><i>Enumerations</i> constrain the values to the list that is specified in this property. For example, you could create a simple type called <i>RainbowColors</i>. You would then add Red, Orange, Yellow, Green, Blue, Indigo, and Violet to the enumerations list.</p> <p>You will need to ensure that you have all variations of the data that you are likely to receive in the message defined in the list. For example, Yellow, yellow, ye1, y could be possible variations of a single color.</p> <p>Select <b>Add</b> to add a default enumeration. Overtyping the default with the data you require.</p> <p>If you need to change an entry, select the entry, then click on the entry a second time (as distinct from double-click). The selected entry can then be updated.</p>

## Patterns

Property	Type	Meaning
Patterns	String	<p><i>Patterns</i> are a regular expression or a series of regular expressions used to constrain the data within the simple type. For further information on patterns and their syntax see "Using regular expressions to parse data elements" on page 695.</p> <p>Select <b>Add</b> to add a default pattern. Overtyping the default with the data you require.</p> <p>If you need to change an entry, select the entry, then click on the entry a second time (as distinct from double-click). The selected entry can then be updated.</p>



Logical properties for value constraints for simple type decimal types:

The simple type value constraint properties described here apply to:

- Objects: Simple types
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger

### Inclusive Constraints

Property	Type	Meaning
Min	Integer	<p>The <i>Min</i> property is used to specify the minimum value for which the data in the message must be greater than or equal to.</p> <p>When this value is set it cannot be equal to or greater than the <i>Max</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Min</i> Inclusive Constraint and <i>Min</i> Exclusive Constraint properties together for the same simple type.</p>
Max	Integer	<p>The <i>Max</i> property is used to specify the maximum value for which the data in the message must be less than or equal to.</p> <p>When this value is set it cannot be equal to or less than the <i>Min</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Max</i> Inclusive Constraint and <i>Max</i> Exclusive Constraint properties together for the same simple type.</p>

### Exclusive Constraints

Property	Type	Meaning
Min	Integer	<p>The <i>Min</i> property is used to specify the minimum value for which the data in the message must be greater than.</p> <p>When this value is set it cannot be equal to or greater than the <i>Max</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Min</i> Inclusive Constraint and <i>Min</i> Exclusive Constraint properties together for the same simple type.</p>
Max	Integer	<p>The <i>Max</i> property is used to specify the maximum value for which the data in the message must be less than.</p> <p>When this value is set it cannot be equal to or less than the <i>Min</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Max</i> Inclusive Constraint and <i>Max</i> Exclusive Constraint properties together for the same simple type.</p>

Property	Type	Meaning
Fraction Digits	Integer	<p>Set this property to limit the number of digits in the fraction part of a numerical value to the number of digits specified in this property.</p> <p>The value must be greater than or equal to 0 and less than 2147483648.</p> <p>The value set for <i>Fraction Digits</i> cannot be greater than the value specified for <i>Total Digits</i>.</p>

Property	Type	Meaning
Total Digits	Integer	<p>Set this property to set the maximum number of digits in a numerical value to the number of digits specified in this property.</p> <p>The value must be greater than or equal to 0 and less than 2147483648.</p> <p>The value set for <i>Total Digits</i> cannot be less than the value specified for <i>Fraction Digits</i>.</p>

Property	Type	Meaning
White Space	Enumerated type	<p>The <i>White Space</i> controls the processing of white space characters received for this type.</p> <p>The value of <i>White Space</i> must be one of;</p> <ul style="list-style-type: none"> <li>• preserve. If you set the property to preserve, all white space characters including carriage return, line feed and tab are preserved.</li> <li>• replace. If you set the property to replace, all carriage return, line feed and tab characters are replaced with a space character.</li> <li>• collapse. If you set the property to collapse, all carriage return, line feed and tab characters are replaced with a space character. Any adjacent white space characters are then collapsed to a single space character and any leading or trailing spaces are stripped from the data.</li> </ul>

## Enumerations

Property	Type	Meaning
Enumerations	String	<p><i>Enumerations</i> constrain the values to the list that is specified in this property. For example, you could create a simple type called <i>RainbowColors</i>. You would then add Red, Orange, Yellow, Green, Blue, Indigo, and Violet to the enumerations list.</p> <p>You will need to ensure that you have all variations of the data that you are likely to receive in the message defined in the list. For example, Yellow, yellow, yel, y could be possible variations of a single color.</p> <p>Select <b>Add</b> to add a default enumeration. Overtyping the default with the data you require.</p> <p>If you need to change an entry, select the entry, then click on the entry a second time (as distinct from double-click). The selected entry can then be updated.</p>

## Patterns

Property	Type	Meaning
Patterns	String	<p><i>Patterns</i> are a regular expression or a series of regular expressions used to constrain the data within the simple type. For further information on patterns and their syntax see "Using regular expressions to parse data elements" on page 695.</p> <p>Select <b>Add</b> to add a default pattern. Overtyping the default with the data you require.</p> <p>If you need to change an entry, select the entry, then click on the entry a second time (as distinct from double-click). The selected entry can then be updated.</p>

Logical properties for value constraints for simple type float types:

The simple type value constraint properties described here apply to:

- Objects: Simple types
- Float schema types: double, float

### Inclusive Constraints

Property	Type	Meaning
Min	Integer	<p>The <i>Min</i> property is used to specify the minimum value for which the data in the message must be greater than or equal to.</p> <p>When this value is set it cannot be equal to or greater than the <i>Max</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Min</i> Inclusive Constraint and <i>Min</i> Exclusive Constraint properties together for the same simple type.</p>
Max	Integer	<p>The <i>Max</i> property is used to specify the maximum value for which the data in the message must be less than or equal to.</p> <p>When this value is set it cannot be equal to or less than the <i>Min</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Max</i> Inclusive Constraint and <i>Max</i> Exclusive Constraint properties together for the same simple type.</p>

### Exclusive Constraints

Property	Type	Meaning
Min	Integer	<p>The <i>Min</i> property is used to specify the minimum value for which the data in the message must be greater than.</p> <p>When this value is set it cannot be equal to or greater than the <i>Max</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Min</i> Inclusive Constraint and <i>Min</i> Exclusive Constraint properties together for the same simple type.</p>
Max	Integer	<p>The <i>Max</i> property is used to specify the maximum value for which the data in the message must be less than.</p> <p>When this value is set it cannot be equal to or less than the <i>Min</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Max</i> Inclusive Constraint and <i>Max</i> Exclusive Constraint properties together for the same simple type.</p>

Property	Type	Meaning
White Space	Enumerated type	<p>The <i>White Space</i> controls the processing of white space characters received for this type.</p> <p>The value of <i>White Space</i> must be one of;</p> <ul style="list-style-type: none"> <li>• preserve. If you set the property to preserve, all white space characters including carriage return, line feed and tab are preserved.</li> <li>• replace. If you set the property to replace, all carriage return, line feed and tab characters are replaced with a space character.</li> <li>• collapse. If you set the property to collapse, all carriage return, line feed and tab characters are replaced with a space character. Any adjacent white space characters are then collapsed to a single space character and any leading or trailing spaces are stripped from the data.</li> </ul>

## Enumerations

Property	Type	Meaning
Enumerations	String	<p><i>Enumerations</i> constrain the values to the list that is specified in this property. For example, you could create a simple type called <i>RainbowColors</i>. You would then add Red, Orange, Yellow, Green, Blue, Indigo, and Violet to the enumerations list.</p> <p>You will need to ensure that you have all variations of the data that you are likely to receive in the message defined in the list. For example, Yellow, yellow, yel, y could be possible variations of a single color.</p> <p>Select <b>Add</b> to add a default enumeration. Overtyping the default with the data you require.</p> <p>If you need to change an entry, select the entry, then click on the entry a second time (as distinct from double-click). The selected entry can then be updated.</p>

## Patterns

Property	Type	Meaning
Patterns	String	<p><i>Patterns</i> are a regular expression or a series of regular expressions used to constrain the data within the simple type. For further information on patterns and their syntax see "Using regular expressions to parse data elements" on page 695.</p> <p>Select <b>Add</b> to add a default pattern. Overtyping the default with the data you require.</p> <p>If you need to change an entry, select the entry, then click on the entry a second time (as distinct from double-click). The selected entry can then be updated.</p>

*Logical properties for value constraints for simple type integer types:*

The simple type value constraint properties described here apply to:

- Objects: Simple types
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedLong, unsignedShort

## Inclusive Constraints

Property	Type	Meaning
Min	Integer	<p>The <i>Min</i> property is used to specify the minimum value for which the data in the message must be greater than or equal to.</p> <p>When this value is set it cannot be equal to or greater than the <i>Max</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Min</i> Inclusive Constraint and <i>Min</i> Exclusive Constraint properties together for the same simple type.</p>
Max	Integer	<p>The <i>Max</i> property is used to specify the maximum value for which the data in the message must be less than or equal to.</p> <p>When this value is set it cannot be equal to or less than the <i>Min</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Max</i> Inclusive Constraint and <i>Max</i> Exclusive Constraint properties together for the same simple type.</p>

## Exclusive Constraints

Property	Type	Meaning
Min	Integer	<p>The <i>Min</i> property is used to specify the minimum value for which the data in the message must be greater than.</p> <p>When this value is set it cannot be equal to or greater than the <i>Max</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Min</i> Inclusive Constraint and <i>Min</i> Exclusive Constraint properties together for the same simple type.</p>
Max	Integer	<p>The <i>Max</i> property is used to specify the maximum value for which the data in the message must be less than.</p> <p>When this value is set it cannot be equal to or less than the <i>Min</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Max</i> Inclusive Constraint and <i>Max</i> Exclusive Constraint properties together for the same simple type.</p>

Property	Type	Meaning
Fraction Digits	Integer	<p>Set this property to limit the number of digits in the fraction part of a numerical value to the number of digits specified in this property.</p> <p>The value must be greater than or equal to 0 and less than 2147483648.</p> <p>The value set for <i>Fraction Digits</i> cannot be greater than the value specified for <i>Total Digits</i>.</p>

Property	Type	Meaning
Total Digits	Integer	<p>Set this property to set the maximum number of digits in a numerical value to the number of digits specified in this property.</p> <p>The value must be greater than or equal to 0 and less than 2147483648.</p> <p>The value set for <i>Total Digits</i> cannot be less than the value specified for <i>Fraction Digits</i>.</p>

Property	Type	Meaning
White Space	Enumerated type	<p>The <i>White Space</i> controls the processing of white space characters received for this type.</p> <p>The value of <i>White Space</i> must be one of;</p> <ul style="list-style-type: none"> <li>• preserve. If you set the property to preserve, all white space characters including carriage return, line feed and tab are preserved.</li> <li>• replace. If you set the property to replace, all carriage return, line feed and tab characters are replaced with a space character.</li> <li>• collapse. If you set the property to collapse, all carriage return, line feed and tab characters are replaced with a space character. Any adjacent white space characters are then collapsed to a single space character and any leading or trailing spaces are stripped from the data.</li> </ul>

## Enumerations

Property	Type	Meaning
Enumerations	String	<p><i>Enumerations</i> constrain the values to the list that is specified in this property. For example, you could create a simple type called <i>RainbowColors</i>. You would then add Red, Orange, Yellow, Green, Blue, Indigo, and Violet to the enumerations list.</p> <p>You will need to ensure that you have all variations of the data that you are likely to receive in the message defined in the list. For example, Yellow, yellow, yel, y could be possible variations of a single color.</p> <p>Select <b>Add</b> to add a default enumeration. Overtyping the default with the data you require.</p> <p>If you need to change an entry, select the entry, then click on the entry a second time (as distinct from double-click). The selected entry can then be updated.</p>

## Patterns

Property	Type	Meaning
Patterns	String	<p><i>Patterns</i> are a regular expression or a series of regular expressions used to constrain the data within the simple type. For further information on patterns and their syntax see "Using regular expressions to parse data elements" on page 695.</p> <p>Select <b>Add</b> to add a default pattern. Overtyping the default with the data you require.</p> <p>If you need to change an entry, select the entry, then click on the entry a second time (as distinct from double-click). The selected entry can then be updated.</p>

*Logical properties for value constraints for simple type interval types:*

The simple type value constraint properties described here apply to:

- Objects: Simple types
- Interval schema types: duration

## Inclusive Constraints

Property	Type	Meaning
Min	Integer	<p>The <i>Min</i> property is used to specify the minimum value for which the data in the message must be greater than or equal to.</p> <p>When this value is set it cannot be equal to or greater than the <i>Max</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Min</i> Inclusive Constraint and <i>Min</i> Exclusive Constraint properties together for the same simple type.</p>
Max	Integer	<p>The <i>Max</i> property is used to specify the maximum value for which the data in the message must be less than or equal to.</p> <p>When this value is set it cannot be equal to or less than the <i>Min</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Max</i> Inclusive Constraint and <i>Max</i> Exclusive Constraint properties together for the same simple type.</p>

## Exclusive Constraints

Property	Type	Meaning
Min	Integer	<p>The <i>Min</i> property is used to specify the minimum value for which the data in the message must be greater than.</p> <p>When this value is set it cannot be equal to or greater than the <i>Max</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Min</i> Inclusive Constraint and <i>Min</i> Exclusive Constraint properties together for the same simple type.</p>
Max	Integer	<p>The <i>Max</i> property is used to specify the maximum value for which the data in the message must be less than.</p> <p>When this value is set it cannot be equal to or less than the <i>Min</i> Inclusive Constraint property.</p> <p>You cannot specify both <i>Max</i> Inclusive Constraint and <i>Max</i> Exclusive Constraint properties together for the same simple type.</p>

Property	Type	Meaning
White Space	Enumerated type	<p>The <i>White Space</i> controls the processing of white space characters received for this type.</p> <p>The value of <i>White Space</i> must be one of;</p> <ul style="list-style-type: none"> <li>• preserve. If you set the property to preserve, all white space characters including carriage return, line feed and tab are preserved.</li> <li>• replace. If you set the property to replace, all carriage return, line feed and tab characters are replaced with a space character.</li> <li>• collapse. If you set the property to collapse, all carriage return, line feed and tab characters are replaced with a space character. Any adjacent white space characters are then collapsed to a single space character and any leading or trailing spaces are stripped from the data.</li> </ul>

## Enumerations

Property	Type	Meaning
Enumerations	String	<p><i>Enumerations</i> constrain the values to the list that is specified in this property. For example, you could create a simple type called <i>RainbowColors</i>. You would then add Red, Orange, Yellow, Green, Blue, Indigo, and Violet to the enumerations list.</p> <p>You will need to ensure that you have all variations of the data that you are likely to receive in the message defined in the list. For example, Yellow, yellow, yel, y could be possible variations of a single color.</p> <p>Select <b>Add</b> to add a default enumeration. Overtyping the default with the data you require.</p> <p>If you need to change an entry, select the entry, then click on the entry a second time (as distinct from double-click). The selected entry can then be updated.</p>

## Patterns

Property	Type	Meaning
Patterns	String	<p><i>Patterns</i> are a regular expression or a series of regular expressions used to constrain the data within the simple type. For further information on patterns and their syntax see “Using regular expressions to parse data elements” on page 695.</p> <p>Select <b>Add</b> to add a default pattern. Overtyping the default with the data you require.</p> <p>If you need to change an entry, select the entry, then click on the entry a second time (as distinct from double-click). The selected entry can then be updated.</p>

*Logical properties for value constraints for simple type string types:*

The simple type value constraint properties described here apply to:

- Objects: Simple types
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedLong, unsignedShort

## Length constraints

Property	Type	Meaning
Length	Integer	<p>The length property is used to specify the exact length of the simple type in bytes or characters.</p> <p>The value must be greater than 0 and less than 2147483648.</p>
Min	Integer	<p>The Min property is used to specify the minimum length of the simple type in bytes or characters.</p> <p>The value must be greater than 0 and less than 2147483648.</p>
Max	Integer	<p>The Max property is used to specify the maximum length of the simple type in bytes or characters.</p> <p>The value must be greater than 0 and less than 2147483648.</p>



Property	Type	Meaning
White Space	Enumerated type	<p>The <i>White Space</i> controls the processing of white space characters received for this type.</p> <p>The value of <i>White Space</i> must be one of;</p> <ul style="list-style-type: none"> <li>• preserve. If you set the property to preserve, all white space characters including carriage return, line feed and tab are preserved.</li> <li>• replace. If you set the property to replace, all carriage return, line feed and tab characters are replaced with a space character.</li> <li>• collapse. If you set the property to collapse, all carriage return, line feed and tab characters are replaced with a space character. Any adjacent white space characters are then collapsed to a single space character and any leading or trailing spaces are stripped from the data.</li> </ul>

## Enumerations

Property	Type	Meaning
Enumerations	String	<p><i>Enumerations</i> constrain the values to the list that is specified in this property. For example, you could create a simple type called <i>RainbowColors</i>. You would then add Red, Orange, Yellow, Green, Blue, Indigo, and Violet to the enumerations list.</p> <p>You will need to ensure that you have all variations of the data that you are likely to receive in the message defined in the list. For example, Yellow, yellow, yel, y could be possible variations of a single color.</p> <p>Select <b>Add</b> to add a default enumeration. Overtyping the default with the data you require.</p> <p>If you need to change an entry, select the entry, then click on the entry a second time (as distinct from double-click). The selected entry can then be updated.</p>

## Patterns

Property	Type	Meaning
Patterns	String	<p><i>Patterns</i> are a regular expression or a series of regular expressions used to constrain the data within the simple type. For further information on patterns and their syntax see "Using regular expressions to parse data elements" on page 695.</p> <p>Select <b>Add</b> to add a default pattern. Overtyping the default with the data you require.</p> <p>If you need to change an entry, select the entry, then click on the entry a second time (as distinct from double-click). The selected entry can then be updated.</p>

### Simple type CWF properties:

There are no properties to show.

### Simple type XML properties:

There are no properties to show.

### Simple type TDS properties:

There are no properties to show.

#### **Documentation properties for all message set objects:**

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### **Note:**

1. Key, Keyref, and Unique objects do not have documentation properties.

#### **Unique properties**

A unique can have the following properties;

- “Unique logical properties” on page 202
- “Unique CWF properties” on page 209
- “Unique XML properties” on page 218
- “Unique TDS properties” on page 233
- “Documentation properties for all message set objects” on page 175

#### **Unique logical properties:**

There are no properties to show.

#### **Unique CWF properties:**

There are no properties to show.

#### **Unique XML properties:**

There are no properties to show.

#### **Unique TDS properties:**

There are no properties to show.

#### **Documentation properties for all message set objects:**

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### **Note:**

1. Key, Keyref, and Unique objects do not have documentation properties.

#### **Wildcard attribute properties**

A wildcard attribute can have the following properties;

- “Wildcard attribute logical properties” on page 202
- “Wildcard attribute CWF properties” on page 210
- “Wildcard attribute XML properties” on page 218
- “Wildcard attribute TDS properties” on page 233

- “Documentation properties for all message set objects” on page 175

**Wildcard attribute logical properties:**

Property	Type	Meaning
Namespace	String	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references.  This field is initially blank.

Property	Type	Meaning
Process Content	Enumerated type	If a message contains an attribute that corresponds to a wildcard in the message model, <i>Process Content</i> defines how the attribute is validated.  Select from; <ul style="list-style-type: none"> <li>• <i>strict</i>. The parser can only match against attributes declared in the specified namespace.</li> <li>• <i>lax</i>. The parser attempts to match against attributes declared in any accessible namespace. If the specified namespace cannot be found, an error will not be generated.</li> <li>• <i>skip</i>. If you select <i>skip</i> the parser does not perform any validation on the attribute.</li> </ul>

**Wildcard attribute CWF properties:**

There are no properties to show.

**Wildcard attribute XML properties:**

There are no properties to show.

**Wildcard attribute TDS properties:**

There are no properties to show.

**Documentation properties for all message set objects:**

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

**Note:**

1. Key, Keyref, and Unique objects do not have documentation properties.

**Wildcard element properties**

A wildcard element can have the following properties;

- “Wildcard element logical properties” on page 202
- “Wildcard element CWF properties” on page 210
- “Wildcard element XML properties” on page 218
- “Wildcard element TDS properties” on page 233
- “Documentation properties for all message set objects” on page 175

### Wildcard element logical properties:

Property	Type	Meaning
Namespace	String	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references.  This field is initially blank.

Property	Type	Meaning
Process Content	Enumerated type	If a message contains an attribute that corresponds to a wildcard in the message model, <i>Process Content</i> defines how the attribute is validated.  Select from; <ul style="list-style-type: none"><li>• <i>strict</i>. The parser can only match against attributes declared in the specified namespace.</li><li>• <i>lax</i>. The parser attempts to match against attributes declared in any accessible namespace. If the specified namespace cannot be found, an error will not be generated.</li><li>• <i>skip</i>. If you select <i>skip</i> the parser does not perform any validation on the attribute.</li></ul>

### Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default is 1.  If the value is set to 0, then the object is optional.  With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i> , it must be less than or equal to the value in <i>Max Occurs</i> .  This property is ignored by brokers that are prior to WebSphere Message Broker Version 6.0.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default is 1.  If this property is not set, then the object can not occur more than once.  If this property is set to 0, it will be interpreted as if the object does not exist in the message.  It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.  This property is ignored by brokers that are prior to WebSphere Message Broker Version 6.0.

### Wildcard element CWF properties:

There are no properties to show.

### Wildcard element XML properties:

There are no properties to show.

### **Wildcard element TDS properties:**

There are no properties to show.

### **Documentation properties for all message set objects:**

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### **Note:**

1. Key, Keyref, and Unique objects do not have documentation properties.

---

## **Deprecated message model object properties**

Some objects in the message model are deprecated. Examples of such deprecation are objects that are only created as a result of migrating a message set from WebSphere MQ Integrator Broker Version 2.1.

There are two ways of accessing the reference information for the properties of deprecated message model objects. The following topics allow you to access the property information by property kind:

- “Logical properties for deprecated message model objects”
- “Physical properties for deprecated message model objects” on page 552
- “Documentation properties for all message set objects” on page 175

Alternatively, you can access the property information by object, starting from the following topic:

- “Deprecated message model object properties by object” on page 556

## **Logical properties for deprecated message model objects**

Logical property information is available for the following deprecated objects:

- “Compound element logical properties” on page 550
- “Embedded simple type logical properties” on page 552

## Compound element logical properties

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"> <li>• - the hyphen</li> <li>• _ the underscore</li> <li>• . the period</li> </ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <i>xml</i> or any variant (for example <i>Xml</i>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>
Namespace	Enumerated type	<p>Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references.</p> <p>If <code>&lt;no target namespace&gt;</code> is displayed, a namespace has not been set for this object.</p> <p>If the property is inactive, the message set has not been configured to support namespaces.</p> <p>Where the property is active, namespaces that are available for selection will be displayed in the drop down list.</p>
Nilable	Check box	Select this if you want the element to be able to be defined as null. This is distinct from being empty where there is no data in the element.
Abstract	Check box	Select this if you do not want the element to appear in the message, but require one of the members of its substitution group to appear in its place.

## Value

Property	Type	Meaning
Default	Button and String	<p>This property provides the default value for the element or attribute.</p> <p>When writing a fixed-length portion of a message (CWF or fixed-length TDS) if the attribute or element is missing from the message tree, the default value will be inserted into the bit stream in order to preserve the message structure.</p> <p>This property is not used by the message broker when parsing.</p> <p>Note that this behavior differs from the XML Schema specification.</p>
Fixed	Button and String	<p>If a <i>Fixed</i> value is set for an attribute and a message is received, if the attribute exists in the message, the data in the attribute property of the message must match that held in the fixed value of the attribute definition.</p> <p>If the attribute does not exist in the message, the broker will create the attribute and populate it with the data held in the fixed value of the attribute definition.</p>

Property	Type	Meaning
Interpret Value As	Enumerated Type	<p>Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur.</p> <p>The possible values for this property are:</p> <ul style="list-style-type: none"> <li>• None This is the default value and will indicate that the element or attribute does not have any key value associated with it.</li> <li>• Message Set Identity. Specifies that the value of the element or attribute will correspond to the identifier, name or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value will remain in force unless a new element or attribute Message Set Identity field is encountered which resets the Message Set Identity value to this new one.</li> <li>• Message Identity. Specifies that the value of the element or attribute will correspond to the name or alias (in that priority order) that is associated with a message and will act as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity will apply until a new element or attribute Message Identity field is encountered at the same level in the tree. The embedded message may be defined in either the current message set or in a message set identified using a Message Set Identity.</li> </ul> <p><b>Note:</b> This property is only applicable when the type of the object is derived from xsd:string.</p>

### Occurrences

Property	Type	Meaning
Min Occurs	Integer	<p>Specify the minimum number of times that the object can repeat. The default is 1.</p> <p>If the value is set to 0, then the object is optional.</p> <p>With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i>, it must be less than or equal to the value in <i>Max Occurs</i>.</p>
Max Occurs	Integer	<p>Specify the maximum number of times that the object can repeat. The default is 1.</p> <p>If this property is not set, then the object can not occur more than once.</p> <p>If this property is set to 0, it will be interpreted as if the object does not exist in the message.</p> <p>It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.</p>

### Compound element complex type logical properties:

Only the complex type properties shown in the tables below are applicable to compound elements.

Property	Type	Meaning
Name	String	This property is set to <b>**ANONYMOUS**</b> and cannot be changed.

*Content:*

Property	Type	Meaning
Group Reference	Button	This radio button is already selected and cannot be changed.
Group Name	Enumerated type	The Group Name is the name of the group that this complex type is referring to. The groups available to be referenced can be selected from the drop down list.

### Compound element value constraint properties:

The properties for compound element value constraints are identical to simple type value constraints. See “Simple type logical value constraints” on page 201 for details.

### Embedded simple type logical properties Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default is 1.  If the value is set to 0, then the object is optional.  With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i> , it must be less than or equal to the value in <i>Max Occurs</i> .
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default is 1.  If this property is not set, then the object can not occur more than once.  If this property is set to 0, it will be interpreted as if the object does not exist in the message.  It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.

## Physical properties for deprecated message model objects

Property information is available for deprecated objects within:

- “Custom Wire Format properties for deprecated message model objects”
- “XML wire format physical properties for deprecated message model objects” on page 553
- “Tagged/delimited string format physical properties for deprecated objects” on page 555

### Custom Wire Format properties for deprecated message model objects

Custom wire format physical property information is available for the following deprecated objects:

- “Compound element CWF properties”
- “Embedded simple type CWF properties” on page 553

### Compound element CWF properties:



The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - ComIbmMrm _BaseValueBinary	Boolean types - ComIbmMrm _BaseValueBoolean	DateTime types - ComIbmMrm _BaseValueDateTime	Decimal types - ComIbmMrm _BaseValueDecimal
Float types - ComIbmMrm _BaseValueFloat	Integer types - ComIbmMrm _BaseValueInt	String types - ComIbmMrm _BaseValueString	

*Compound element complex type CWF properties:*

There are no properties to show.

**Embedded simple type CWF properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - ComIbmMrm _AnonBinary	Boolean types - ComIbmMrm _AnonBoolean	DateTime types - ComIbmMrm _AnonDate - ComIbmMrm _AnonDateTime - ComIbmMrm _AnonGDay - ComIbmMrm _AnonGMonth - ComIbmMrm _AnonGMonthDay - ComIbmMrm _AnonGYear - ComIbmMrm _AnonGYearMonth - ComIbmMrm _AnonTime	Decimal types - ComIbmMrm _AnonDecimal
Float types - ComIbmMrm _AnonFloat	Integer types - ComIbmMrm _AnonInt	String types - ComIbmMrm _AnonString	

**XML wire format physical properties for deprecated message model objects**

XML wire format physical property information is available for the following deprecated objects:

- “Compound element XML properties”
- “Embedded simple type XML properties”

**Compound element XML properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - ComIbmMrm _BaseValueBinary	Boolean types - ComIbmMrm _BaseValueBoolean	DateTime types - ComIbmMrm _BaseValueDateTime	Decimal types - ComIbmMrm _BaseValueDecimal
Float types - ComIbmMrm _BaseValueFloat	Integer types - ComIbmMrm _BaseValueInt	String types - ComIbmMrm _BaseValueString	

*Compound element complex type XML properties:*

There are no properties to show.

**Embedded simple type XML properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - ComIbmMrm _AnonBinary	Boolean types - ComIbmMrm _AnonBoolean	DateTime types - ComIbmMrm _AnonDate - ComIbmMrm _AnonDateTime - ComIbmMrm _AnonGDay - ComIbmMrm _AnonGMonth - ComIbmMrm _AnonGMonthDay - ComIbmMrm _AnonGYear - ComIbmMrm _AnonGYearMonth - ComIbmMrm _AnonTime	Decimal types - ComIbmMrm _AnonDecimal
Float types - ComIbmMrm _AnonFloat	Integer types - ComIbmMrm _AnonInt	String types - ComIbmMrm _AnonString	

## Tagged/delimited string format physical properties for deprecated objects

TDS format physical property information is available for the following deprecated objects:

- “Compound element TDS properties”
- “Embedded simple type TDS properties”

### Compound element TDS properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - ComIbmMrm _BaseValueBinary	Boolean types - ComIbmMrm _BaseValueBoolean	DateTime types - ComIbmMrm _BaseValueDateTime	Decimal types - ComIbmMrm _BaseValueDecimal
Float types - ComIbmMrm _BaseValueFloat	Integer types - ComIbmMrm _BaseValueInt	String types - ComIbmMrm _BaseValueString	

### Compound element complex type TDS properties:

The TDS properties for compound element complex types are identical to the TDS properties for normal complex types. See “Complex type TDS properties” on page 219 for details.

### Embedded simple type TDS properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - ComIbmMrm _AnonBinary	Boolean types - ComIbmMrm _AnonBoolean	DateTime types - ComIbmMrm _AnonDate - ComIbmMrm _AnonDateTime - ComIbmMrm _AnonGDay - ComIbmMrm _AnonGMonth - ComIbmMrm _AnonGMonthDay - ComIbmMrm _AnonGYear - ComIbmMrm _AnonGYearMonth - ComIbmMrm _AnonTime	Decimal types - ComIbmMrm _AnonDecimal
--	--	--	--

Float types - ComIbmMrm _AnonFloat	Integer types - ComIbmMrm _AnonInt	String types - ComIbmMrm _AnonString	
--	--	--	--

## Documentation properties for all message set objects

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

### Note:

1. Key, Keyref, and Unique objects do not have documentation properties.

## Deprecated message model object properties by object

The following deprecated objects have properties that can be viewed or set:

- “Compound element properties”
- “Embedded simple type properties” on page 621

### Compound element properties

A compound element can have the following properties;

- “Compound element logical properties” on page 550
- “Compound element CWF properties” on page 552
- “Compound element XML properties” on page 554
- “Compound element TDS properties” on page 555
- “Documentation properties for all message set objects” on page 175

### Compound element logical properties:

Property	Type	Meaning
Name	String	<p>Specify a name for the object when you create it.</p> <p>Names can consist of virtually any alphanumeric character including the letters <i>A</i> through <i>Z</i>, <i>a</i> through <i>z</i> and the digits <i>0</i> through <i>9</i>.</p> <p>They may also include the following punctuation characters;</p> <ul style="list-style-type: none"> <li>• - the hyphen</li> <li>• _ the underscore</li> <li>• . the period</li> </ul> <p>Names can only start with a letter or the underscore character and not with a number, hyphen or period.</p> <p>Names beginning with <code>xml</code> or any variant (for example <code>XML</code>) are reserved by the XML standards specification.</p> <p>Further details of naming conventions and allowable characters can be found in the Extensible Markup Language (XML) specification that can be found on the World Wide Web Consortium (W3C) Web site.</p>

Property	Type	Meaning
Namespace	Enumerated type	Namespaces are a simple method for qualifying element and attribute names by associating them with namespaces identified by URI references.  If <no target namespace> is displayed, a namespace has not been set for this object.  If the property is inactive, the message set has not been configured to support namespaces.  Where the property is active, namespaces that are available for selection will be displayed in the drop down list.
Nilable	Check box	Select this if you want the element to be able to be defined as null. This is distinct from being empty where there is no data in the element.
Abstract	Check box	Select this if you do not want the element to appear in the message, but require one of the members of its substitution group to appear in its place.

### Value

Property	Type	Meaning
Default	Button and String	This property provides the default value for the element or attribute.  When writing a fixed-length portion of a message (CWF or fixed-length TDS) if the attribute or element is missing from the message tree, the default value will be inserted into the bit stream in order to preserve the message structure.  This property is not used by the message broker when parsing.  Note that this behavior differs from the XML Schema specification.
Fixed	Button and String	If a <i>Fixed</i> value is set for an attribute and a message is received, if the attribute exists in the message, the data in the attribute property of the message must match that held in the fixed value of the attribute definition.  If the attribute does not exist in the message, the broker will create the attribute and populate it with the data held in the fixed value of the attribute definition.
Interpret Value As	Enumerated Type	Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur.  The possible values for this property are: <ul style="list-style-type: none"> <li>• None This is the default value and will indicate that the element or attribute does not have any key value associated with it.</li> <li>• Message Set Identity. Specifies that the value of the element or attribute will correspond to the identifier, name or alias (in that priority order) that is associated with the message set where all subsequent embedded messages that are descendents of the enclosing message are defined. This value will remain in force unless a new element or attribute Message Set Identity field is encountered which resets the Message Set Identity value to this new one.</li> <li>• Message Identity. Specifies that the value of the element or attribute will correspond to the name or alias (in that priority order) that is associated with a message and will act as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity will apply until a new element or attribute Message Identity field is encountered at the same level in the tree. The embedded message may be defined in either the current message set or in a message set identified using a Message Set Identity.</li> </ul> <p><b>Note:</b> This property is only applicable when the type of the object is derived from xsd:string.</p>

## Occurrences

Property	Type	Meaning
Min Occurs	Integer	<p>Specify the minimum number of times that the object can repeat. The default is 1.</p> <p>If the value is set to 0, then the object is optional.</p> <p>With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i>, it must be less than or equal to the value in <i>Max Occurs</i>.</p>
Max Occurs	Integer	<p>Specify the maximum number of times that the object can repeat. The default is 1.</p> <p>If this property is not set, then the object can not occur more than once.</p> <p>If this property is set to 0, it will be interpreted as if the object does not exist in the message.</p> <p>It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.</p>

### *Compound element value constraint properties:*

The properties for compound element value constraints are identical to simple type value constraints. See “Simple type logical value constraints” on page 201 for details.

### **Compound element CWF properties:**

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - ComIbmMrm _BaseValueBinary	Boolean types - ComIbmMrm _BaseValueBoolean	DateTime types - ComIbmMrm _BaseValueDateTime	Decimal types - ComIbmMrm _BaseValueDecimal
Float types - ComIbmMrm _BaseValueFloat	Integer types - ComIbmMrm _BaseValueInt	String types - ComIbmMrm _BaseValueString	

### *CWF properties for compound element binary types:*

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

## Physical representation

Property	Type	Meaning
Length Count	Button and Integer	<p>If you have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 1.</p> <p>The maximum value that you can specify is 2147483647.</p> <p>The default value is empty (not set).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message’s CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message’s CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>

## Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.  Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.  If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.

*CWF properties for compound element boolean types:*

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

## Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	Specify how the object is aligned from the start of the message. Select one of: <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>



Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.  Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.  If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.

*CWF properties for compound element dateTime types:*

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>Fixed Length String. The element's length is determined by other length properties below.</li> <li>Length Encoded String 1. The element's first byte contains the length of the string following the length byte in length units. The maximum length of a Length Encoded String 1 element is 255 length units.</li> <li>Length Encoded String 2. The element's first two bytes contain the length of the string following the 2 length bytes in length units. The maximum length of a Length Encoded String 2 element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding.</li> <li>Null Terminated String. The string ends with the hexadecimal NULL character, 'X'00'.</li> <li>Packed Decimal. The date<code>Time</code> is coded as a Packed Decimal number. It is valid only if the <i>Date<code>Time</code> Format</i> property represents numeric-only data.</li> <li>Binary. The date<code>Time</code> is encoded as a binary sequence of bytes. If you select this option, the range of symbols that you can specify for the Format String property is less than the range of symbols you can specify if you select a string option (see "Date<code>Time</code> formats" on page 701 for details).</li> <li>Time Seconds. This value supports C <code>time_t</code> and Java Date and Time objects. It is valid only if the <i>Date<code>Time</code> Format</i> property represents numeric-only data.</li> <li>Time Milliseconds. This value supports C <code>time_t</code> and Java Date and Time objects. It is valid only if the <i>Date<code>Time</code> Format</i> property represents numeric-only data.</li> </ul> <p>The default value is fixed length string.</p>
Date <code>Time</code> Format	String	<p>Specify a template for date and time.</p> <p>The default date<code>Time</code> format is dependent on the logical type of the object. For information on the defaults for the date<code>Time</code> format according to the logical type see "Date<code>Time</code> defaults by logical type" on page 709.</p> <p>If you set the <i>Physical Type</i> to Binary, the template is restricted to those components defined in "Date<code>Time</code> as string data" on page 701. If you set the <i>Physical Type</i> to Packed Decimal, Time Seconds, or Time Milliseconds, the template is restricted to those components that represent numbers. In these cases, you must update this <i>Date<code>Time</code> Format</i> property.</p> <p>See "Date<code>Time</code> formats" on page 701 for details of date and time formats.</p>
Length Count	Button and Integer	<p>If you have selected a <i>Physical Type</i> of Fixed Length String, Packed Decimal, or Binary, and have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 1 for all three physical types.</p> <p>The maximum value that you can specify is 256 for Fixed Length String, 10 for Packed Decimal, and 2147483647 for Binary.</p> <p>The default value is 0 (zero).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	<p>If you have set the <i>Physical Type</i> property to Packed Decimal, Time Seconds, or Time Milliseconds, select (the default) or unselect <i>Signed</i>. If you have selected another value for <i>Physical Type</i>, this property is invalid.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to Fixed Length String, select Left Justify (the default value) or Right Justify from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Fixed Length String, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>• <b>NULLPadFill</b>. This is only valid if <i>Physical Type</i> is Fixed Length String. The field is filled with the value specified by the <i>Padding Character</i>. This is the default value.</li> <li>• <b>NULLLogicalValue</b>. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>• <b>NULLLiteralValue</b>. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. Use this option when the value you have set for <i>Encoding Null Value</i> to specify a null date is not a date<code>Time</code> value, or does not conform to the standard date<code>Time</code> format <code>yyyy-MM-dd 'T'HH:mm:ss</code>.</li> <li>• <b>NULLLiteralFill</b>. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>
Encoding Null Value	String	<p>If you set the <i>Encoding Null</i> property to <b>NULLPadFill</b>, this property is disabled (grayed out).</p> <p>If you set the <i>Encoding Null</i> property to <b>NULLLogicalValue</b>, you must set this property to an ISO8601 date<code>Time</code> format. These formats are described in “Date<code>Time</code> as string data” on page 701. For example, specify a value conforming to <code>yyyy-MM-dd'T'HH:mm:ss</code> such as <code>1970-12-01</code>.</p> <p>If you set the <i>Encoding Null</i> property to <b>NULLLiteralValue</b>, you can enter any value that is the same length as the field.</p> <p>If you set the <i>Encoding Null</i> property to <b>NULLLiteralFill</b>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <b>SPACE</b>, <b>NUL</b>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <code>c</code> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <code>YY</code> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <code>YY</code> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <code>xxxx</code> is a Unicode value specified in hexadecimal format.</li> </ul>

## Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.  Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.  If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.

*CWF properties for compound element decimal types:*

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	Select one of the following from the drop-down list: <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer and Packed Decimal are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>

Property	Type	Meaning
Length Count	Integer	<p>Enter the number of bytes to specify the element length:</p> <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you set the <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 10.</li> <li>• If you set the <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)</li> </ul>
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	<p>Select (the default) or deselect this property. This property is used in conjunction with <i>Sign Orientation</i>.</p>
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to External Decimal and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• <b>Included Leading.</b> This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• <b>Separate Leading.</b> This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• <b>Included Trailing.</b> This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• <b>Separate Trailing.</b> This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
Virtual Decimal Point	Integer	<p>Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point should be moved from its assumed position. For example, a decimal element containing 1234 with a Virtual Decimal value of 3 is 1.234. This is equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>



Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is External Decimal. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string. You can specify a nonnumeric value for <i>Encoding Null Value</i>.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>

Property	Type	Meaning
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above, with the exception of <code>NULLLiteralFill</code>. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <i>c</i> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <i>YY</i> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <i>YY</i> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <i>xxxx</i> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

*CWF properties for compound element float types:*

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Float. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL. This is the default value.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer, Packed Decimal, and Float are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	<p>Enter the number of bytes to specify the element length:</p> <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Float, select a value from the drop-down list. The default value is 8.</li> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you set the <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 10.</li> <li>• If you set the <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)</li> </ul>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	<p>Select or deselect (unsigned, the default) this property. If you have set <i>Physical Type</i> to <i>Float</i>, this is selected. This property is used in conjunction with <i>Sign Orientation</i>.</p>
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to <i>External Decimal</i> and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to <i>Leading</i> or <i>Trailing</i> (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• <b>Included Leading.</b> This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• <b>Separate Leading.</b> This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• <b>Included Trailing.</b> This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• <b>Separate Trailing.</b> This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
Virtual Decimal Point	Integer	<p>Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point should be moved from its assumed position. For example, a float element containing 1234 with a Virtual Decimal value of 3 is 1.234.</p> <p>This is not applicable if you have set <i>Physical Type</i> to Float.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is External Decimal. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string. You can specify a nonnumeric value for <i>Encoding Null Value</i>.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>

Property	Type	Meaning
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above, with the exception of <code>NULLLiteralFill</code>. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <code>c</code> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <code>YY</code> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <code>YY</code> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <code>xxxx</code> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

*CWF properties for compound element integer types:*

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer and Packed Decimal are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	<p>Enter the number of bytes to specify the element length:</p> <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you have set <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 6.</li> <li>• If you have set <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 11.</li> </ul>



Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	Select (the default) or deselect this property. This property is used in conjunction with <i>Sign Orientation</i> .
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to External Decimal and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• <b>Included Leading.</b> This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• <b>Separate Leading.</b> This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• <b>Included Trailing.</b> This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• <b>Separate Trailing.</b> This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is External Decimal. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string. You can specify a nonnumeric value for <i>Encoding Null Value</i>.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>

Property	Type	Meaning
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above, with the exception of <code>NULLLiteralFill</code>. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <i>c</i> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <i>YY</i> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <i>YY</i> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <i>xxxx</i> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

*CWF properties for compound element string types:*

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Fixed Length String. The element's length is determined by other length properties below.</li> <li>• Length Encoded String 1. The element's first byte contains the length of the string following the length byte in length units. The maximum length of a Length Encoded String 1 element is 255 length units.</li> <li>• Length Encoded String 2. The element's first two bytes contain the length of the string following the 2 length bytes in length units. The maximum length of a Length Encoded String 2 element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding.</li> <li>• Null Terminated String. The string ends with the hexadecimal NULL character, X'00'.</li> </ul> <p>The default is Fixed Length String.</p>
Length Count	Button and Integer	<p>If you have selected a <i>Physical Type</i> of Fixed Length String or Binary, and have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 0 (zero), the maximum value that you can specify is 2147483647</p> <p>The default value is 0 (zero).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to Fixed Length String, select Left Justify (the default value) or Right Justify from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Fixed Length String, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is Fixed Length String. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>

Property	Type	Meaning
Encoding Null Value	STRING	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. If specified, its length must be equal to the length of the string element, with the exception of <code>NULLLiteralFill</code>.</p> <p>The default value is empty (not set).</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <code>c</code> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <code>YY</code> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <code>YY</code> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <code>xxxx</code> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>



## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

### Compound element XML properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - ComIbmMrm _BaseValueBinary	Boolean types - ComIbmMrm _BaseValueBoolean	DateTime types - ComIbmMrm _BaseValueDateTime	Decimal types - ComIbmMrm _BaseValueDecimal
Float types - ComIbmMrm _BaseValueFloat	Integer types - ComIbmMrm _BaseValueInt	String types - ComIbmMrm _BaseValueString	

*XML wire format properties for compound element binary types:*

The XML Wire Format properties described here apply to:

- Objects: Compound elements

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is <i>id</i>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <i>val</i>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

## Physical representation

Property	Type	Meaning
Encoding	String	<p>Select one of the following values from the drop-down list: :</p> <ul style="list-style-type: none"> <li>• CDatahex (the default)</li> <li>• hex</li> <li>• base64</li> </ul>

*XML wire format properties for compound element boolean types:*

The XML Wire Format properties described here apply to:

- Objects: Compound elements

## Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li> <p><b>XMLElement.</b> If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element.</p> <p>If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element.</p> <p>This is the default value for element objects.</p> </li> <li> <p><b>XMLAttribute.</b> If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements.</p> <p>If you select this value for more than one object, you must set their <i>XML Name</i> property to different values.</p> <p>This is the default value for attribute objects.</p> </li> <li> <p><b>XMLElementAttrID.</b> If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li> <p><b>XMLElementAttrVal.</b> If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</p> </li> <li> <p><b>XMLElementAttrIDVal.</b> This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML wire format properties for compound element dateTime types:*

The XML Wire Format properties described here apply to:

- Objects: Compound elements

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li> <p><b>XMLElement.</b> If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element.</p> <p>If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element.</p> <p>This is the default value for element objects.</p> </li> <li> <p><b>XMLAttribute.</b> If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements.</p> <p>If you select this value for more than one object, you must set their <i>XML Name</i> property to different values.</p> <p>This is the default value for attribute objects.</p> </li> <li> <p><b>XMLElementAttrID.</b> If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li> <p><b>XMLElementAttrVal.</b> If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</p> </li> <li> <p><b>XMLElementAttrIDVal.</b> This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

## Physical representation

Property	Type	Meaning
DateTime Format	String	<p>Specify a format string that specifies the rendering of the value for <code>dateTime</code> elements.</p> <p>The default <code>dateTime</code> format is dependent on the logical type of the object. For information on the defaults for the <code>dateTime</code> format according to the logical type see "DateTime defaults by logical type" on page 709.</p> <p>See "DateTime formats" on page 701 for details of <code>dateTime</code> formats.</p>

*XML wire format properties for compound element decimal types:*

The XML Wire Format properties described here apply to:

- Objects: Compound elements

## Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li> <p><b>XMLElement.</b> If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element.</p> <p>If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element.</p> <p>This is the default value for element objects.</p> </li> <li> <p><b>XMLAttribute.</b> If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements.</p> <p>If you select this value for more than one object, you must set their <i>XML Name</i> property to different values.</p> <p>This is the default value for attribute objects.</p> </li> <li> <p><b>XMLElementAttrID.</b> If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li> <p><b>XMLElementAttrVal.</b> If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</p> </li> <li> <p><b>XMLElementAttrIDVal.</b> This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>



Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML wire format properties for compound element float types:*

The XML Wire Format properties described here apply to:

- Objects: Compound elements

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li> <p><b>XMLElement.</b> If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element.</p> <p>If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element.</p> <p>This is the default value for element objects.</p> </li> <li> <p><b>XMLAttribute.</b> If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements.</p> <p>If you select this value for more than one object, you must set their <i>XML Name</i> property to different values.</p> <p>This is the default value for attribute objects.</p> </li> <li> <p><b>XMLElementAttrID.</b> If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li> <p><b>XMLElementAttrVal.</b> If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</p> </li> <li> <p><b>XMLElementAttrIDVal.</b> This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML wire format properties for compound element integer types:*

The XML Wire Format properties described here apply to:

- Objects: Compound elements

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li> <p><b>XMLElement.</b> If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element.</p> <p>If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element.</p> <p>This is the default value for element objects.</p> </li> <li> <p><b>XMLAttribute.</b> If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements.</p> <p>If you select this value for more than one object, you must set their <i>XML Name</i> property to different values.</p> <p>This is the default value for attribute objects.</p> </li> <li> <p><b>XMLElementAttrID.</b> If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li> <p><b>XMLElementAttrVal.</b> If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</p> </li> <li> <p><b>XMLElementAttrIDVal.</b> This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>.</p> <p>If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values:</p> <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is <code>id</code>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <code>XMLElement</code>, <code>XMLAttribute</code>, or <code>XMLElementAttrVal</code>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <code>val</code>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

*XML wire format properties for compound element string types:*

The XML Wire Format properties described here apply to:

- Objects: Compound elements

### Field identification

A number of the following properties will only become active depending on the value that *Render* property is set to.

Property	Type	Meaning
Render	Enumerated Type	<p>Specify how the instantiated object or type is rendered (output) in the resulting XML document. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>XMLElement</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the content of the child element. If you select this value for more than one object, and set their <i>XML Name</i> property to the same value, both objects must refer to the same element. This is the default value for element objects.</li> <li>• <b>XMLAttribute</b>. If you select this value, the object (or type) is rendered as an attribute of the parent XML object. The identity of the child is determined by the attribute name. The value is the attribute value. This is only valid for simple elements. If you select this value for more than one object, you must set their <i>XML Name</i> property to different values. This is the default value for attribute objects.</li> <li>• <b>XMLElementAttrID</b>. If you select this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the value of a specified attribute of the child. The value is the content of the child element. You must add an attribute to the child element with an attribute name as specified in <i>ID Attribute Name</i> and a value as specified in <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrIDVal</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> <li>• <b>XMLElementAttrVal</b>. If you specify this value, the object (or type) is rendered as a child XML element of the parent complex type. The identity of the child is determined by the tag name of the child. The value is the value of a specified attribute. The name of the attribute is specified in <i>Value Attribute Name</i>.</li> <li>• <b>XMLElementAttrIDVal</b>. This option combines the two options, <i>XMLElementAttrID</i> and <i>XMLElementAttrVal</i>. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of <i>ID Attribute Name</i>. The value is the value of <i>ID Attribute Value</i>. If you select this value for one object, and set this same value or the value <i>XMLElementAttrID</i> for a second object, and set <i>XML Name, ID Attribute Name, ID Attribute Value</i> to the same values: <ul style="list-style-type: none"> <li>– You must also set <i>Value Attribute Name</i> to the same value for the two objects.</li> <li>– Both objects must refer to the same element.</li> </ul> </li> </ul> <p>“XML rendering options” on page 678 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.</p>

Property	Type	Meaning
XML Name	String	<p>Enter a value for the XML element name. This property specifies the name for the XML start tag or attribute for the element (or attribute) in an XML document (message).</p> <p>This can be used to provide name mapping when the MRM identifier needs to be different from the XML name, for example because of different namespace rules. It must be a valid XML name.</p> <p>You cannot specify a name that is already used for another element (or attribute) , or for a message. No two elements (or attribute) or messages can have the same XML name.</p> <p>If you do not set a value, it defaults to that of the element's identifier. If the element's identifier is a prefixed identifier, it defaults to the identifier with the caret character (^) replaced by an underscore (_).</p>
ID Attribute Name	String	<p>Specify the name of the attribute used to identify the child. This must be a valid XML Attribute Name. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is <i>id</i>.</p>
Namespace	String	Enter the namespace associated with the <i>ID Attribute</i> .
ID Attribute Value	String	<p>Specify the value of the attribute used to identify the child. This property is ignored and cannot be changed (the field is disabled) if <i>Render</i> is set to <i>XMLElement</i>, <i>XMLAttribute</i>, or <i>XMLElementAttrVal</i>.</p> <p>The default value is the identifier of the child.</p>
Value Attribute Name	String	<p>Specify the name of the attribute used for the value of the child. This must be a valid XML Attribute Name. This is only used if required by the setting of <i>Render</i>.</p> <p>The default value is <i>val</i>.</p>
Namespace	String	Enter the namespace associated with the <i>Value Attribute</i> .

### Compound element TDS properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - ComIbmMrm _BaseValueBinary	Boolean types - ComIbmMrm _BaseValueBoolean	DateTime types - ComIbmMrm _BaseValueDateTime	Decimal types - ComIbmMrm _BaseValueDecimal
Float types - ComIbmMrm _BaseValueFloat	Integer types - ComIbmMrm _BaseValueInt	String types - ComIbmMrm _BaseValueString	

*TDS properties for compound element binary types:*

The TDS properties described here apply to:

- Objects: Compound elements

## Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

## Physical representation

Property	Type	Meaning
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p> <p>Regardless of the value of the <i>Data Element Separator</i> property, either the <i>Length</i> or <i>Length Reference</i> property must be set.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

*TDS properties for compound element boolean types:*



The TDS properties described here apply to:

- Objects: Compound elements

### Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the <i>Data Element Separation</i> of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

### Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>

Property	Type	Meaning
Length Reference	Enumerated type	Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.  For information about reordering elements, see “Reordering objects” on page 106.

*TDS properties for compound element datetime types:*

The TDS properties described here apply to:

- Objects: Compound elements

### Field identification

Property	Type	Meaning
Tag	String	Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.  Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.  The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.
Data Pattern	String	Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.
Repeating Element Delimiter	String	Specify the delimiter to be used between repeating elements.  This delimiter is only used if the element repeats and if the <i>Data Element Separation</i> of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.  A default value is provided if the above conditions are true as follows: <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> If none of the above rules are met, a default will not be applied.

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
DateTime Format	String	<p>Specify a template for date and time.</p> <p>The default DateTime format is dependent on the logical type of the object. For information on the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 709.</p> <p>See "DateTime formats" on page 701 for details of date and time formats.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see “TDS Null handling options” on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a dateTime object to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

*TDS properties for compound element decimal types:*

The TDS properties described here apply to:

- Objects: Compound elements

## Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.</p>

Property	Type	Meaning
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

### Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

## Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	<p>Specify a nonzero integer representing the position of an implied decimal point within a number, or specify 0 (zero, the default) to use the formatting of Float or Decimal numbers as specified by the <i>Precision</i> property.</p> <p>If you specify a positive integer, the position of the decimal point is moved left from the right hand side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234</p> <p>If you specify a negative integer, the position of the decimal point is moved right from the right hand side of the number. For example, if you specify -3, the decimal value 1234 represents 1234000</p>
Precision	Button and Integer	<p>This value is used if the <i>Virtual Decimal Point</i> property value is 0, indicating that the decimal point is present in the data. It deals with truncation, and specifies the number of digits that should follow the decimal point.</p> <p>Either specify a number of digits:</p> <ul style="list-style-type: none"> <li>• If you set <i>Precision</i> to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123.</li> <li>• If you set <i>Precision</i> to a number less than the number of fractional digits, data is truncated. For example, the value 123.4567 is truncated to 123.45 if you set <i>Precision</i> to 2.</li> <li>• If you set <i>Precision</i> to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value 12.345 is padded to 12.34500 if you set <i>Precision</i> to 5.</li> </ul> <p>Or select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• All Significant Digits (the default): all the significant digits are written to the output bit stream, and there is no decimal separator if there are no fractional digits.</li> <li>• Explicit Decimal Separator: all the significant digits are written to the output bit stream and the decimal separator is always included, even if there are no fractional digits. The decimal separator must be present in the input bitstream, even if no fractional digits are present.</li> </ul>
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	<p>Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.</p>
Negative Sign	String	<p>Specify the value that represents the negative symbol. If no value is set, "-" is assumed.</p>



## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see “TDS Null handling options” on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a dateTime object to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in “Date Time as string data” on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

*TDS properties for compound element float types:*

The TDS properties described here apply to:

- Objects: Compound elements

## Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.</p>

Property	Type	Meaning
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

### Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

## Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	<p>Specify a nonzero integer representing the position of an implied decimal point within a number, or specify 0 (zero, the default) to use the formatting of Float or Decimal numbers as specified by the <i>Precision</i> property.</p> <p>If you specify a positive integer, the position of the decimal point is moved left from the right hand side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234</p> <p>If you specify a negative integer, the position of the decimal point is moved right from the right hand side of the number. For example, if you specify -3, the decimal value 1234 represents 1234000</p>
Precision	Button and Integer	<p>This value is used if the <i>Virtual Decimal Point</i> property value is 0, indicating that the decimal point is present in the data. It deals with truncation, and specifies the number of digits that should follow the decimal point.</p> <p>Either specify a number of digits:</p> <ul style="list-style-type: none"> <li>• If you set <i>Precision</i> to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123.</li> <li>• If you set <i>Precision</i> to a number less than the number of fractional digits, data is truncated. For example, the value 123.4567 is truncated to 123.45 if you set <i>Precision</i> to 2.</li> <li>• If you set <i>Precision</i> to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value 12.345 is padded to 12.34500 if you set <i>Precision</i> to 5.</li> </ul> <p>Or select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• All Significant Digits (the default): all the significant digits are written to the output bit stream, and there is no decimal separator if there are no fractional digits.</li> <li>• Explicit Decimal Separator: all the significant digits are written to the output bit stream and the decimal separator is always included, even if there are no fractional digits. The decimal separator must be present in the input bitstream, even if no fractional digits are present.</li> <li>• Exponential Notation: data is written out to the bit stream as a signed value having the format [sign1]a.bb[sign2]ccc where: <ul style="list-style-type: none"> <li>– [sign1] is the value of Negative Sign if the value is negative</li> <li>– a is a single decimal digit</li> <li>– bb is one or more decimal digits</li> <li>– [sign2] is the value of Negative Sign if the exponent is negative</li> <li>– ccc is exactly three decimal digits (the exponent)</li> </ul> </li> </ul> <p>[sign1] and [sign2] are absent if the value and exponent, respectively, are positive.</p> <p>For example, the value -123.456 is represented as -1.23456e002 and the value 0.00012 is represented as 1.2e-004 in the output bit stream (assuming the value of <i>Negative Sign</i> is "-" and <i>Sign Orientation</i> is Leading).</p> <p>The value -0.00012 is represented as 1.2*e*004 if <i>Negative Sign</i> is "*" and <i>Sign Orientation</i> is Trailing.</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

### Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>• NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For date<code>Time</code> elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see "TDS Null handling options" on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a date<code>Time</code> object to NULLLogicalValue, you must set this property to an ISO8601 date<code>Time</code> format. These formats are described in "Date<code>Time</code> as string data" on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

*TDS properties for compound element integer types:*

The TDS properties described here apply to:

- Objects: Compound elements

## Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the <i>Data Element Separation</i> of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>

Property	Type	Meaning
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p> <p>Regardless of the value of the <i>Data Element Separator</i> property, either the <i>Length</i> or <i>Length Reference</i> property must be set.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>



## Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>• NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see "TDS Null handling options" on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a dateTime object to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

*TDS properties for compound element string types:*

The TDS properties described here apply to:

- Objects: Compound elements

## Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>
Interpret Element Value	Enumerated Type	<p>Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. This interpretation is generally standard-specific and is therefore hard coded.</p> <p>The possible values for this property are:</p> <ul style="list-style-type: none"> <li>• None (the default value)</li> <li>• EDIFACT Service String</li> <li>• X12 Service String</li> <li>• Message Key</li> <li>• EDIFACT Syntax Level ID</li> <li>• HL7 Service String</li> <li>• HL7 Field Separator</li> </ul>
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the <i>Data Element Separation</i> of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see “TDS Null handling options” on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a dateTime object to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

### Documentation properties for all message set objects:

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### Note:

1. Key, Keyref, and Unique objects do not have documentation properties.

### Embedded simple type properties

An embedded simple type can have the following properties;

- “Embedded simple type logical properties” on page 552
- “Embedded simple type CWF properties” on page 553
- “Embedded simple type XML properties” on page 554
- “Embedded simple type TDS properties” on page 555
- “Documentation properties for all message set objects” on page 175

### Embedded simple type logical properties:

#### Occurrences

Property	Type	Meaning
Min Occurs	Integer	<p>Specify the minimum number of times that the object can repeat. The default is 1.</p> <p>If the value is set to 0, then the object is optional.</p> <p>With the exception of <i>Max Occurs</i> being set to -1, if a value is set for <i>Min Occurs</i>, it must be less than or equal to the value in <i>Max Occurs</i>.</p>

Property	Type	Meaning
Max Occurs	Integer	<p>Specify the maximum number of times that the object can repeat. The default is 1.</p> <p>If this property is not set, then the object can not occur more than once.</p> <p>If this property is set to 0, it will be interpreted as if the object does not exist in the message.</p> <p>It can also be set to -1 to indicate that the limit is unbounded and there is no maximum to the number of occurrences.</p>

### Embedded simple type CWF properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - ComIbmMrm _AnonBinary	Boolean types - ComIbmMrm _AnonBoolean	DateTime types - ComIbmMrm _AnonDate - ComIbmMrm _AnonDateTime - ComIbmMrm _AnonGDay - ComIbmMrm _AnonGMonth - ComIbmMrm _AnonGMonthDay - ComIbmMrm _AnonGYear - ComIbmMrm _AnonGYearMonth - ComIbmMrm _AnonTime	Decimal types - ComIbmMrm _AnonDecimal
Float types - ComIbmMrm _AnonFloat	Integer types - ComIbmMrm _AnonInt	String types - ComIbmMrm _AnonString	

*CWF properties for embedded simple type binary types:*

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

## Physical representation

Property	Type	Meaning
Length Count	Button and Integer	<p>If you have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 1.</p> <p>The maximum value that you can specify is 2147483647.</p> <p>The default value is empty (not set).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see “Reordering objects” on page 106.</p>
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message’s CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message’s CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>

## Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.  Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.  If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.

*CWF properties for embedded simple type boolean types:*

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

## Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	Specify how the object is aligned from the start of the message. Select one of: <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>



Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

### Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.  Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.  If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.

*CWF properties for embedded simple type dateTime types:*

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>Fixed Length String. The element's length is determined by other length properties below.</li> <li>Length Encoded String 1. The element's first byte contains the length of the string following the length byte in length units. The maximum length of a Length Encoded String 1 element is 255 length units.</li> <li>Length Encoded String 2. The element's first two bytes contain the length of the string following the 2 length bytes in length units. The maximum length of a Length Encoded String 2 element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding.</li> <li>Null Terminated String. The string ends with the hexadecimal NULL character, X'00'.</li> <li>Packed Decimal. The date<code>Time</code> is coded as a Packed Decimal number. It is valid only if the <i>Date<code>Time</code> Format</i> property represents numeric-only data.</li> <li>Binary. The date<code>Time</code> is encoded as a binary sequence of bytes. If you select this option, the range of symbols that you can specify for the Format String property is less than the range of symbols you can specify if you select a string option (see "Date<code>Time</code> formats" on page 701 for details).</li> <li>Time Seconds. This value supports C <code>time_t</code> and Java Date and Time objects. It is valid only if the <i>Date<code>Time</code> Format</i> property represents numeric-only data.</li> <li>Time Milliseconds. This value supports C <code>time_t</code> and Java Date and Time objects. It is valid only if the <i>Date<code>Time</code> Format</i> property represents numeric-only data.</li> </ul> <p>The default value is fixed length string.</p>
Date <code>Time</code> Format	String	<p>Specify a template for date and time.</p> <p>The default date<code>Time</code> format is dependent on the logical type of the object. For information on the defaults for the date<code>Time</code> format according to the logical type see "Date<code>Time</code> defaults by logical type" on page 709.</p> <p>If you set the <i>Physical Type</i> to Binary, the template is restricted to those components defined in "Date<code>Time</code> as string data" on page 701. If you set the <i>Physical Type</i> to Packed Decimal, Time Seconds, or Time Milliseconds, the template is restricted to those components that represent numbers. In these cases, you must update this <i>Date<code>Time</code> Format</i> property.</p> <p>See "Date<code>Time</code> formats" on page 701 for details of date and time formats.</p>
Length Count	Button and Integer	<p>If you have selected a <i>Physical Type</i> of Fixed Length String, Packed Decimal, or Binary, and have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 1 for all three physical types.</p> <p>The maximum value that you can specify is 256 for Fixed Length String, 10 for Packed Decimal, and 2147483647 for Binary.</p> <p>The default value is 0 (zero).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	<p>If you have set the <i>Physical Type</i> property to Packed Decimal, Time Seconds, or Time Milliseconds, select (the default) or unselect <i>Signed</i>. If you have selected another value for <i>Physical Type</i>, this property is invalid.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to Fixed Length String, select Left Justify (the default value) or Right Justify from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Fixed Length String, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>• <b>NULLPadFill</b>. This is only valid if <i>Physical Type</i> is Fixed Length String. The field is filled with the value specified by the <i>Padding Character</i>. This is the default value.</li> <li>• <b>NULLLogicalValue</b>. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>• <b>NULLLiteralValue</b>. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. Use this option when the value you have set for <i>Encoding Null Value</i> to specify a null date is not a date<code>Time</code> value, or does not conform to the standard date<code>Time</code> format <code>yyyy-MM-dd 'T'HH:mm:ss</code>.</li> <li>• <b>NULLLiteralFill</b>. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>
Encoding Null Value	String	<p>If you set the <i>Encoding Null</i> property to <b>NULLPadFill</b>, this property is disabled (grayed out).</p> <p>If you set the <i>Encoding Null</i> property to <b>NULLLogicalValue</b>, you must set this property to an ISO8601 date<code>Time</code> format. These formats are described in “Date<code>Time</code> as string data” on page 701. For example, specify a value conforming to <code>yyyy-MM-dd'T'HH:mm:ss</code> such as <code>1970-12-01</code>.</p> <p>If you set the <i>Encoding Null</i> property to <b>NULLLiteralValue</b>, you can enter any value that is the same length as the field.</p> <p>If you set the <i>Encoding Null</i> property to <b>NULLLiteralFill</b>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <b>SPACE</b>, <b>NUL</b>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <i>c</i> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <i>YY</i> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <i>YY</i> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <i>xxxx</i> is a Unicode value specified in hexadecimal format.</li> </ul>

## Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>

Property	Type	Meaning
Leading Skip Count	Integer	Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to the first instance only.
Trailing Skip Count	Integer	Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set Byte Alignment Pad property.  For repeating objects, this property is applied to all instances.

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.  Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.  If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.

*CWF properties for embedded simple type decimal types:*

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	Select one of the following from the drop-down list: <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer and Packed Decimal are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>

Property	Type	Meaning
Length Count	Integer	<p>Enter the number of bytes to specify the element length:</p> <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you set the <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 10.</li> <li>• If you set the <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)</li> </ul>
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	<p>Select (the default) or deselect this property. This property is used in conjunction with <i>Sign Orientation</i>.</p>
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to External Decimal and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• <b>Included Leading.</b> This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• <b>Separate Leading.</b> This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• <b>Included Trailing.</b> This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• <b>Separate Trailing.</b> This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
Virtual Decimal Point	Integer	<p>Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point should be moved from its assumed position. For example, a decimal element containing 1234 with a Virtual Decimal value of 3 is 1.234. This is equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>



Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is External Decimal. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string. You can specify a nonnumeric value for <i>Encoding Null Value</i>.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>

Property	Type	Meaning
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above, with the exception of <code>NULLLiteralFill</code>. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <code>c</code> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <code>YY</code> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <code>YY</code> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <code>xxxx</code> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

*CWF properties for embedded simple type float types:*

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Float. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL. This is the default value.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer, Packed Decimal, and Float are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	<p>Enter the number of bytes to specify the element length:</p> <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Float, select a value from the drop-down list. The default value is 8.</li> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you set the <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 10.</li> <li>• If you set the <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 256. (Numbers greater than the maximum COBOL PICTURE clause of 18 are assumed to be 18.)</li> </ul>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	<p>Select or deselect (unsigned, the default) this property. If you have set <i>Physical Type</i> to <i>Float</i>, this is selected. This property is used in conjunction with <i>Sign Orientation</i>.</p>
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to <i>External Decimal</i> and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to <i>Leading</i> or <i>Trailing</i> (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• <b>Included Leading.</b> This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• <b>Separate Leading.</b> This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• <b>Included Trailing.</b> This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• <b>Separate Trailing.</b> This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
Virtual Decimal Point	Integer	<p>Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point should be moved from its assumed position. For example, a float element containing 1234 with a Virtual Decimal value of 3 is 1.234.</p> <p>This is not applicable if you have set <i>Physical Type</i> to Float.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is External Decimal. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string. You can specify a nonnumeric value for <i>Encoding Null Value</i>.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>

Property	Type	Meaning
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above, with the exception of <code>NULLLiteralFill</code>. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <code>c</code> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <code>YY</code> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <code>YY</code> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <code>xxxx</code> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

*CWF properties for embedded simple type integer types:*

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Integer. This equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL.</li> <li>• Packed Decimal. This equates to the COMP-3 data type in COBOL.</li> <li>• External Decimal. This equates to the data type PIC 9 USAGE DISPLAY in COBOL.</li> </ul> <p>The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager:</p> <ul style="list-style-type: none"> <li>• Elements that have <i>Physical Type</i> set to Integer and Packed Decimal are represented in the appropriate WebSphere MQ Encoding value.</li> <li>• Elements that have <i>Physical Type</i> set to External Decimal are represented in the WebSphere MQ CodedCharSetId value.</li> </ul>
Length Count	Integer	<p>Enter the number of bytes to specify the element length:</p> <ul style="list-style-type: none"> <li>• If you set the <i>Physical Type</i> to Integer, select 1, 2, or 4 (the default) from the drop-down list.</li> <li>• If you have set <i>Physical Type</i> to Packed Decimal, enter a value between 1 and 6.</li> <li>• If you have set <i>Physical Type</i> to Extended Decimal, enter a value between 1 and 11.</li> </ul>



Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i>.</li> </ul> <p>The default is bytes.</p>
Signed	Boolean	Select (the default) or deselect this property. This property is used in conjunction with <i>Sign Orientation</i> .
Signed EBCDIC Custom	Boolean	<p>If the <i>Physical Type</i> is set to External Decimal and the <i>Signed EBCDIC Custom</i> property is set, this will indicate that the Sign EBCDIC Custom representation is to be used within an ASCII environment. If this check box is not set (the default) then the Sign ASCII representation will be used.</p> <p>The setting of the <i>Sign EBCDIC Custom</i> check box is only appropriate if the <i>Sign Orientation</i> property is set to Leading or Trailing (indicating that the element/attribute has an embedded sign representation).</p> <p>The check box will be disabled if the element/attribute is unsigned (for example, where the <i>Signed</i> check box is not set).</p>

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>If you have set <i>Physical Type</i> to External Decimal and you have selected <i>Signed</i>, choose from the following options that represent the COBOL options for displaying numeric data:</p> <ul style="list-style-type: none"> <li>• <b>Included Leading.</b> This sets a bit in the first byte on if the number is negative. No setting is made if the number is positive. For example, the ASCII hexadecimal representation of the number 22 is x'3232'. Using this option, the number +22 would be x'3232' and the number -22 would be x'7232'. This is the default value.</li> <li>• <b>Separate Leading.</b> This sets the first byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> <li>• <b>Included Trailing.</b> This sets a bit in the last byte on if the number is negative. No setting is made if the number is positive. Using this option, the number +22 would be x'3232' and the number -22 would be x'3272'.</li> <li>• <b>Separate Trailing.</b> This sets the last byte of the element to '+' if the number is positive and to '-' if the number is negative. For this option, the length must include the sign byte.</li> </ul> <p>If you have set <i>Physical Type</i> to any other value, the value Not Applicable is set for you.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to External Decimal, select Left Justify or Right Justify (the default value) from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Extended Decimal, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is External Decimal. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string. You can specify a nonnumeric value for <i>Encoding Null Value</i>.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>

Property	Type	Meaning
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above, with the exception of <code>NULLLiteralFill</code>. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <i>c</i> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <i>YY</i> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <i>YY</i> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <i>xxxx</i> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>

## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

*CWF properties for embedded simple type string types:*

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated Type	<p>Select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Fixed Length String. The element's length is determined by other length properties below.</li> <li>• Length Encoded String 1. The element's first byte contains the length of the string following the length byte in length units. The maximum length of a Length Encoded String 1 element is 255 length units.</li> <li>• Length Encoded String 2. The element's first two bytes contain the length of the string following the 2 length bytes in length units. The maximum length of a Length Encoded String 2 element is 65535 length units. The two length bytes are in the format of the WebSphere MQ queue manager Encoding.</li> <li>• Null Terminated String. The string ends with the hexadecimal NULL character, X'00'.</li> </ul> <p>The default is Fixed Length String.</p>
Length Count	Button and Integer	<p>If you have selected a <i>Physical Type</i> of Fixed Length String or Binary, and have set <i>Length Type</i> to Count, enter the number of length units for the element.</p> <p>The minimum value that you can specify is 0 (zero), the maximum value that you can specify is 2147483647</p> <p>The default value is 0 (zero).</p>
Length Reference	Button and Enumerated Type	<p>If you have selected the length to be defined by <i>Length Reference</i>, select the name of the Integer object that specifies the length of this object. Make your selection from the drop-down list of Integer objects that are defined as siblings of the current object, and occur before it in the structure of the message.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

Property	Type	Meaning
Length Units	Enumerated Type	<p>Subject to the <i>Physical Type</i> that has been set, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Bytes. This specifies that X bytes are processed where X is the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• Characters. The meaning of this option depends on the value that you have set in the message's CCSID or that you have set for the message set property <i>Default CCSID</i>. <ul style="list-style-type: none"> <li>– If you have specified an SBCS CCSID, X bytes are processed where X is the value of <i>Length Count</i> or of the INTEGER specified by <i>Length Reference</i>.</li> <li>– If you have specified a DBCS CCSID, Y bytes are processed, where Y is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i> multiplied by two.</li> <li>– If you have specified an MBCS CCSID, the parser reads 1 character at a time and determines whether the character comprises 1 or more bytes. The parser performs Z reads, where Z is the value of <i>Length Count</i> or of the INTEGER specified in <i>Length Reference</i>.</li> </ul> </li> <li>• Character Units. This specifies that the size of character unit is determined by the value that you have set in the message's CCSID, or for the message set property <i>Default CCSID</i>. The number of bytes processed is the size of character unit multiplied by the value of <i>Length Count</i>, or of the INTEGER specified in <i>Length Reference</i>.</li> <li>• End of Bitstream. All data until end of bit stream is processed. This option is valid only if the element is the last in the message. If you select this value, you do not need to enter a value in <i>Length Count</i> or <i>Length Reference</i>.</li> </ul> <p>The default is bytes.</p>
String Justification	Enumerated Type	<p>If you have set the <i>Physical Type</i> property to Fixed Length String, select Left Justify (the default value) or Right Justify from the drop-down list. If you have selected another value for <i>Physical Type</i>, this property is inactive.</p>

Property	Type	Meaning
Padding Character	String	<p>The padding character is used to fill out the remaining character positions when the string length is less than the specified string size. If you have set the <i>Physical Type</i> property to Fixed Length String, and the <i>String Justification</i> property is either Left Justify or Right Justify, set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid if <i>Physical Type</i> is Fixed Length String. The field is filled with the value specified by the <i>Padding Character</i>. <i>Encoding Null Value</i> must be set to an empty string.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> is transformed to match the required format for the field. This is the default value.</li> <li>• NULLLiteralValue. The <i>Encoding Null Value</i> is directly substituted as if it is a string.</li> <li>• NULLLiteralFill. The field is filled with the value specified by the <i>Encoding Null Value</i>. <i>Encoding Null Value</i> must resolve to a single character.</li> </ul>

Property	Type	Meaning
Encoding Null Value	STRING	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. If specified, its length must be equal to the length of the string element, with the exception of <code>NULLLiteralFill</code>.</p> <p>The default value is empty (not set).</p> <p>If you set the <i>Encoding Null</i> property to <code>NULLLiteralFill</code>, the value must resolve to a single character. Set the character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select <code>SPACE</code>, <code>NUL</code>, <code>0x00</code> or <code>0xFF</code> from the drop-down list</li> <li>• Enter a character between quotation marks, for example <code>'c'</code> or <code>"c"</code>, where <code>c</code> is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form <code>0xYY</code> where <code>YY</code> is a hexadecimal value.</li> <li>• Enter a decimal character code in the form <code>YY</code> where <code>YY</code> is a decimal value.</li> <li>• Enter a Unicode value in the form <code>U+xxxx</code> where <code>xxxx</code> is a Unicode value specified in hexadecimal format.</li> </ul>

### Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated Type	<p>Specify how the object is aligned from the start of the message. Select one of:</p> <ul style="list-style-type: none"> <li>• 1 Bytes. This is the default value.</li> <li>• 2 Bytes</li> <li>• 4 Bytes</li> <li>• 8 Bytes</li> <li>• 16 Bytes</li> </ul>
Leading Skip Count	Integer	<p>Specify the number of bytes to skip before reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a field defined by C or COBOL data which requires alignment on a 2, 4, 8 or 16 byte boundary. Specify the number of bytes to skip before reading or writing this object. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to the first instance only.</p>
Trailing Skip Count	Integer	<p>Specify the number of bytes to skip after reading or writing this object. The default is 0, the minimum value is 0, and the maximum value is 999999. You can use this value to ignore unwanted fields in a structure, or to model a repeating structure containing fields which require alignment on a 2, 4, 8 or 16 byte boundary. When an output message is written, Skip Count bytes are assigned the value of the message set <code>Byte Alignment Pad</code> property.</p> <p>For repeating objects, this property is applied to all instances.</p>



## Repeat

Property	Type	Meaning
Repeat Reference	Enumerated Type	<p>Use this property if the object occurs multiple times, and the number of occurrences is given dynamically by a field earlier in the message.</p> <p>Select an integer object from the drop-down list of integer objects that occur before this object in the structure of the message. The value of the selected integer specifies the number of occurrences of this object. If there are no objects listed, then there are no integer objects prior to this one in the message structure.</p> <p>If a <i>Repeat Reference</i> is specified, it will override any setting for the <i>Max Occurs</i> logical property when parsing and writing the message, but not for validation of the message.</p>

### Embedded simple type XML properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - ComIbmMrm _AnonBinary	Boolean types - ComIbmMrm _AnonBoolean	DateTime types - ComIbmMrm _AnonDate - ComIbmMrm _AnonDateTime - ComIbmMrm _AnonGDay - ComIbmMrm _AnonGMonth - ComIbmMrm _AnonGMonthDay - ComIbmMrm _AnonGYear - ComIbmMrm _AnonGYearMonth - ComIbmMrm _AnonTime	Decimal types - ComIbmMrm _AnonDecimal
Float types - ComIbmMrm _AnonFloat	Integer types - ComIbmMrm _AnonInt	String types - ComIbmMrm _AnonString	

*XML Wire Format properties for embedded simple type binary types:*

The XML wire format properties described here apply to:

- Objects: Embedded simple types

## Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : <ul style="list-style-type: none"><li>• CDatahex (the default)</li><li>• hex</li><li>• base64</li></ul>

*XML wire format properties for embedded simple type boolean types:*

The XML wire format properties described here apply to:

- Objects: Embedded simple types

There are no properties to show.

*XML wire format properties for embedded simple type dateTime types:*

The XML wire format properties described here apply to:

- Objects: Embedded simple types

## Physical representation

Property	Type	Meaning
DateTime Format	String	Specify a format string that specifies the rendering of the value for dateTime elements.  The default dateTime format is dependent on the logical type of the object. For information on the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 709.  See "DateTime formats" on page 701 for details of dateTime formats.

*XML wire format properties for embedded simple type decimal types:*

The XML Wire Format properties described here apply to:

- Objects: Embedded simple types

There are no properties to show.

*XML wire format properties for embedded simple type float types:*

The XML wire format properties described here apply to:

- Objects: Embedded simple types

There are no properties to show.

*XML wire format properties for embedded simple type integer types:*

The XML wire format properties described here apply to:

- Objects: Embedded simple types

There are no properties to show.

*XML wire format properties for embedded simple type string types:*

The XML wire format properties described here apply to:

- Objects: Embedded simple types

There are no properties to show.

### Embedded simple type TDS properties:

The properties displayed on the object page and the values that those properties can take, can vary according to the type of the object. For example, the properties for type string are different to those of type boolean. Select the link for the object type from the table below.

Binary types - ComIbmMrm _AnonBinary	Boolean types - ComIbmMrm _AnonBoolean	DateTime types - ComIbmMrm _AnonDate - ComIbmMrm _AnonDateTime - ComIbmMrm _AnonGDay - ComIbmMrm _AnonGMonth - ComIbmMrm _AnonGMonthDay - ComIbmMrm _AnonGYear - ComIbmMrm _AnonGYearMonth - ComIbmMrm _AnonTime	Decimal types - ComIbmMrm _AnonDecimal
Float types - ComIbmMrm _AnonFloat	Integer types - ComIbmMrm _AnonInt	String types - ComIbmMrm _AnonString	

*TDS properties for embedded simple type binary types:*

The TDS properties described here apply to:

- Objects: Embedded simple types

### Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>

Property	Type	Meaning
Data Pattern	String	Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.
Repeating Element Delimiter	String	Specify the delimiter to be used between repeating elements.  This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.  A default value is provided if the above conditions are true as follows: <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> If none of the above rules are met, a default will not be applied.

### Physical representation

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).  This property applies to simple objects and to complex elements with a base type.  If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.  If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.  Regardless of the value of the <i>Data Element Separator</i> property, either the <i>Length</i> or <i>Length Reference</i> property must be set.
Length Reference	Enumerated type	Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.  For information about reordering elements, see “Reordering objects” on page 106.

*TDS properties for embedded simple type boolean types:*

The TDS properties described here apply to:

- Objects: Embedded simple types

## Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the <i>Data Element Separation</i> of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

*TDS properties for embedded simple type dateTime types:*

The TDS properties described here apply to:

- Objects: Embedded simple types

### Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.</p>
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the <i>Data Element Separation</i> of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

### Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>

Property	Type	Meaning
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
DateTime Format	String	<p>Specify a template for date and time.</p> <p>The default DateTime format is dependent on the logical type of the object. For information on the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 709.</p> <p>See "DateTime formats" on page 701 for details of date and time formats.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>



## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see “TDS Null handling options” on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a dateTime object to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

*TDS properties for embedded simple type decimal types:*

The TDS properties described here apply to:

- Objects: Embedded simple types

## Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.</p>

Property	Type	Meaning
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

### Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

## Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	<p>Specify a nonzero integer representing the position of an implied decimal point within a number, or specify 0 (zero, the default) to use the formatting of Float or Decimal numbers as specified by the <i>Precision</i> property.</p> <p>If you specify a positive integer, the position of the decimal point is moved left from the right hand side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234</p> <p>If you specify a negative integer, the position of the decimal point is moved right from the right hand side of the number. For example, if you specify -3, the decimal value 1234 represents 1234000</p>
Precision	Button and Integer	<p>This value is used if the <i>Virtual Decimal Point</i> property value is 0, indicating that the decimal point is present in the data. It deals with truncation, and specifies the number of digits that should follow the decimal point.</p> <p>Either specify a number of digits:</p> <ul style="list-style-type: none"> <li>• If you set <i>Precision</i> to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123.</li> <li>• If you set <i>Precision</i> to a number less than the number of fractional digits, data is truncated. For example, the value 123.4567 is truncated to 123.45 if you set <i>Precision</i> to 2.</li> <li>• If you set <i>Precision</i> to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value 12.345 is padded to 12.34500 if you set <i>Precision</i> to 5.</li> </ul> <p>Or select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• All Significant Digits (the default): all the significant digits are written to the output bit stream, and there is no decimal separator if there are no fractional digits.</li> <li>• Explicit Decimal Separator: all the significant digits are written to the output bit stream and the decimal separator is always included, even if there are no fractional digits. The decimal separator must be present in the input bitstream, even if no fractional digits are present.</li> </ul>
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	<p>Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.</p>
Negative Sign	String	<p>Specify the value that represents the negative symbol. If no value is set, "-" is assumed.</p>

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see “TDS Null handling options” on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a dateTime object to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

*TDS properties for embedded simple type float types:*

The TDS properties described here apply to:

- Objects: Embedded simple types

## Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See “Regular expression syntax” on page 697 for further details.</p>

Property	Type	Meaning
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the Data Element Separation of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

### Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

## Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	<p>Specify a nonzero integer representing the position of an implied decimal point within a number, or specify 0 (zero, the default) to use the formatting of Float or Decimal numbers as specified by the <i>Precision</i> property.</p> <p>If you specify a positive integer, the position of the decimal point is moved left from the right hand side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234</p> <p>If you specify a negative integer, the position of the decimal point is moved right from the right hand side of the number. For example, if you specify -3, the decimal value 1234 represents 1234000</p>
Precision	Button and Integer	<p>This value is used if the <i>Virtual Decimal Point</i> property value is 0, indicating that the decimal point is present in the data. It deals with truncation, and specifies the number of digits that should follow the decimal point.</p> <p>Either specify a number of digits:</p> <ul style="list-style-type: none"> <li>• If you set <i>Precision</i> to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123.</li> <li>• If you set <i>Precision</i> to a number less than the number of fractional digits, data is truncated. For example, the value 123.4567 is truncated to 123.45 if you set <i>Precision</i> to 2.</li> <li>• If you set <i>Precision</i> to a number greater than the number of fractional digits, the value is padded with extra zeros. For example, the value 12.345 is padded to 12.34500 if you set <i>Precision</i> to 5.</li> </ul> <p>Or select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> <li>• All Significant Digits (the default): all the significant digits are written to the output bit stream, and there is no decimal separator if there are no fractional digits.</li> <li>• Explicit Decimal Separator: all the significant digits are written to the output bit stream and the decimal separator is always included, even if there are no fractional digits. The decimal separator must be present in the input bitstream, even if no fractional digits are present.</li> <li>• Exponential Notation: data is written out to the bit stream as a signed value having the format [sign1]a.bbbe[sign2]ccc where: <ul style="list-style-type: none"> <li>– [sign1] is the value of Negative Sign if the value is negative</li> <li>– a is a single decimal digit</li> <li>– bbb is one or more decimal digits</li> <li>– [sign2] is the value of Negative Sign if the exponent is negative</li> <li>– ccc is exactly three decimal digits (the exponent)</li> </ul> </li> </ul> <p>[sign1] and [sign2] are absent if the value and exponent, respectively, are positive.</p> <p>For example, the value -123.456 is represented as -1.23456e002 and the value 0.00012 is represented as 1.2e-004 in the output bit stream (assuming the value of <i>Negative Sign</i> is "-" and <i>Sign Orientation</i> is Leading).</p> <p>The value -0.00012 is represented as 1.2*e*004 if <i>Negative Sign</i> is "*" and <i>Sign Orientation</i> is Trailing.</p>



Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

### Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>• NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see "TDS Null handling options" on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a dateTime object to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

*TDS properties for embedded simple type integer types:*

The TDS properties described here apply to:

- Objects: Embedded simple types

## Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the <i>Data Element Separation</i> of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>

Property	Type	Meaning
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>

## Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated Type	<p>Select the value that indicates the positioning of a sign symbol within a numeric value, from the drop-down list:</p> <ul style="list-style-type: none"> <li>• None</li> <li>• Leading</li> <li>• Trailing</li> </ul> <p>If you set the value for this property to None, this is interpreted as having no sign, and an exception is thrown if a negative number is processed (on either input or output).</p> <p>If you set the value for this property Leading, this indicates that the sign is positioned ahead of the number, for example, -1234. Similarly, if you set this property to Trailing, the sign follows the number, for example, 1234-.</p> <p>If there is no explicit sign set, the number is assumed to be positive.</p>
Positive Sign	String	Specify the value that represents the positive symbol. If no value is set, "+" is assumed. The positive sign is not written when creating an output message, it is only used to recognize the positive sign when parsing a message bitstream.
Negative Sign	String	Specify the value that represents the negative symbol. If no value is set, "-" is assumed.

## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>• NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>• NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>• NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see "TDS Null handling options" on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a dateTime object to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

*TDS properties for embedded simple type string types:*

The TDS properties described here apply to:

- Objects: Embedded simple types

## Field identification

Property	Type	Meaning
Tag	String	<p>Specify the value used to identify the object in a message bit stream. If the object is simple and the <i>Data Element Separation</i> property of the complex type or types in which the object is a child is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, this property must contain a non-empty value.</p> <p>Should the object be a complex element and the <i>Data Element Separation</i> property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length then the property can contain an empty value.</p> <p>The value for this property must be unique for every element in the message set, that is, no two elements in the message set can contain the same value for this property.</p>
Data Pattern	String	<p>Specify the regular expression to be used by the parser to identify the data in the message to be assigned to the object. Used when the <i>Data Element Separation</i> method has been set to Use Data Pattern in the complex type. See "Regular expression syntax" on page 697 for further details.</p>
Interpret Element Value	Enumerated Type	<p>Specify if values stored within this object must be interpreted as having significance for the parser and, if so, the type of interpretation that must occur. This interpretation is generally standard-specific and is therefore hard coded.</p> <p>The possible values for this property are:</p> <ul style="list-style-type: none"> <li>• None (the default value)</li> <li>• EDIFACT Service String</li> <li>• X12 Service String</li> <li>• Message Key</li> <li>• EDIFACT Syntax Level ID</li> <li>• HL7 Service String</li> <li>• HL7 Field Separator</li> </ul>
Repeating Element Delimiter	String	<p>Specify the delimiter to be used between repeating elements.</p> <p>This delimiter is only used if the element repeats and if the <i>Data Element Separation</i> of the parent group or complex type is set to All Elements Delimited or Variable Length Elements Delimited.</p> <p>A default value is provided if the above conditions are true as follows:</p> <ul style="list-style-type: none"> <li>• If the messaging standard is HL7, the mnemonic &lt;HL7_RS&gt; is used.</li> <li>• If the messaging standard is not HL7, and the maximum number of repeats is fixed, the delimiter of the parent group or complex type is used.</li> </ul> <p>If none of the above rules are met, a default will not be applied.</p>

## Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	<p>The <i>Physical Type</i> can be set to Characters and Messaging Standard Alternate. This property tells the TDS parser whether the data in the message is the normal TDS character format, or is another alternate form that has a specific messaging standard such as TLOG. The available values and the default value depend on both <i>Messaging Standard</i> and the logical type.</p> <ul style="list-style-type: none"> <li>• Where the logical type of the object is of a dateTime, float, or integer type, this property is disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to other than TLOG, the <i>Physical Type</i> property will be disabled.</li> <li>• Where the <i>Messaging Standard</i> property (at the message set level) is set to TLOG and the logical type of the object is set to a boolean, decimal or string type, the <i>Physical Type</i> property can be set to Characters or Messaging Standard Alternate.</li> </ul>
Length	Integer	<p>Specify the expected length of the object in characters (except in the case of binary objects, in which case the length value represents the length in bytes).</p> <p>This property applies to simple objects and to complex elements with a base type.</p> <p>If you give this property a value of 0, the <i>Length Reference</i> property is checked for a value.</p> <p>If you set the <i>Data Element Separator</i> property for the type to Fixed Length or Fixed Length AL3, either this property, or the <i>Length Reference</i> property, must contain a non 0 (or non NULL) value.</p>
Justification	Enumerated Type	<p>Specify the justification of the object where the data being written or parsed is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• Not Applicable</li> <li>• Left Justify</li> <li>• Right Justify</li> </ul>

Property	Type	Meaning
Padding Character	String	<p>Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed length object where the data is less than the fixed length value. This property is only used when a value is output as a fixed length string.</p> <p>Set this character in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Select NUL, '0', or SPACE from the drop-down list.</li> <li>• Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.</li> <li>• Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.</li> <li>• Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal. The maximum length of the string you can enter is 10.</li> </ul> <p>The choice of which of these padding character forms is used for an MRM element will depend on the padding character required and whether the padding character is to be subject to data conversion. In most cases the specification of a padding character will be sufficient and when this padding character is used, it will be converted to the target codepage of the output MRM message being generated. If a padding character is required that cannot easily be entered in the padding character field, then the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value will also be converted to the target codepage of the MRM message being generated. If a padding character is required that is not subject to data conversion, then the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly in the output message. If this format is used then you must ensure that this hexadecimal is valid for the codepage of any output messages that are created using these MRM definitions.</p> <p>If you are converting a message from one code page to another, then you must ensure that the converted value of the padding character is valid for this codepage. For example, when converting from ASCII to the codepage 500, if you have specified the numeric 8 as your padding character then this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'.</p> <p>There is a currently a restriction that the value of your padding character should not be greater than U+007F. You should note that if you enter a Unicode mnemonic or numeric value, it is considered to be the character represented by that number in UTF-8.</p>
Length Reference	Enumerated type	<p>Specify the identifier of a sibling Integer object, the value of which dictates the length of the object in question. The sibling object must be defined before the current object within the message structure.</p> <p>For information about reordering elements, see "Reordering objects" on page 106.</p>



## Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated Type	<p>Select one of the following options from the drop-down list. The option that you select determines the value that you must set for the property <i>Encoding Null Value</i>:</p> <ul style="list-style-type: none"> <li>NULLPadFill. This is only valid for fixed length objects. This is the default value.</li> <li>NULLLogicalValue. The <i>Encoding Null Value</i> property is first converted to an actual value, and rendered in the way specified for the field.</li> <li>NULLLiteralValue. This specifies that <i>Encoding Null Value</i> contains a value that is directly substituted as if it is a string. For dateTime elements, use this option if you want to use the <i>Encoding Null Value</i> to test or compare the content of the field in the message.</li> </ul> <p>For full information about using these options, see “TDS Null handling options” on page 691.</p>
Encoding Null Value	String	<p>The use of this property depends on the <i>Encoding Null</i> property, described above. The default value is zero.</p> <p>If you set the <i>Encoding Null</i> property for a dateTime object to NULLLogicalValue, you must set this property to an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 701. For example, specify a value conforming to yyyy-MM-dd'T'HH:mm:ss such as 1970-12-01.</p>

### Documentation properties for all message set objects:

The documentation property of an object<sup>1</sup> is where you can add information to enhance the understanding of that objects function.

It is a string field and any standard alphanumeric characters can be used.

#### Note:

1. Key, Keyref, and Unique objects do not have documentation properties.

---

## Additional MRM domain information

This section provides additional information in relation to the MRM domain. This information is categorized into:

- “Additional logical information”
- “Additional CWF information” on page 675
- “Additional XML information” on page 676
- “Additional TDS information” on page 679
- “DateTime formats” on page 701

## Additional logical information

This section provides additional information in relation to the logical model. This information has been categorized into:

- “MRM model restrictions” on page 674

## MRM model restrictions

This topic provides information on areas where the WebSphere MQ Integrator Broker does not exactly follow the XML Schema specification.

### XML Schema features supported only in the message editor:

The following features can be created and edited using the message editor, but are not honored by WebSphere Message Broker:

- Pattern facet on non-string data types. The message broker will only validate pattern facets which are applied to simple types based on `xsd:string`.
- Whitespace facet. The message broker does not make use of the whitespace facet at all (although whitespace facets can be included in the message model if necessary). It is possible to accurately control the processing of whitespace using the settings on the physical formats.
- ID attribute. The message model can contain attributes with the name 'id', but these will not be checked for uniqueness.

### XML Schema exceptions:

The following features may be created and edited using the message editor, but the message broker will process them in a way which differs from the XML Schema specification, as follows:

- Default and fixed values. The processing of default and fixed values depends on the physical format in which the message is parsed. For details on how each physical format uses these fields, refer to the concept topic Relationship to the logical model for the relevant physical format.
- `xsi:type` attribute. The `xsi:type` attribute is not automatically processed by the message broker. An attribute with the name 'xsi:type' can be included in the message model, and can be processed using a message flow.

### Differences in validation:

If validation is enabled in a message flow, the following features/scenarios will not be validated in exactly the same way as a validating XML parser would validate them:

- Any Element/Any Attribute. If the message model contains a wildcard ('any element' or 'any attribute') the message broker will validate the 'processContents' field as follows:
  - `skip`. No checking will be done, any element/attribute will be allowed.
  - `lax`. No checking will be done, any element/attribute will be allowed.
  - `strict`. Any element/attribute in the same message set will be allowed.

**Note:** If all of the definitions for a namespace are included within the same message set, the validation of 'strict' will be the same as a validating XML parser.

- Element substitution and 'all' groups. If an element is substitutable, and it occurs within an 'all' group, the following exceptions apply to the validation of the element:
  - A substitutable element within an 'all' group will always be validated as if it were optional.
  - An input message will not be rejected if more than one of the substitutions is used in the same 'all' group.

## Additional CWF information

This section provides additional information in relation to the CWF physical format. This information has been categorized into:

- “CWF data conversion”
- “CWF Null handling options”

### CWF data conversion

You can convert an MRM message to a different code page or encoding, or both. To do this, you should set the `CodedCharSetId` and `Encoding` fields in the appropriate output WebSphere MQ header to the target value. The appropriate WebSphere MQ header is the header that precedes and is adjacent to the output message body.

The data conversion performed is dependent on the simple type of each element:

- Binary schema types: `base64Binary`, `hexBinary` objects are not converted.
- Boolean schema types: `boolean` objects are not converted.
- DateTime schema types: `date`, `dateTime`, `gDay`, `gMonth`, `gMonthDay`, `gYear`, `gYearMonth`, `time` objects are handled as either binary, string, or packed decimals. If a `dateTime` element is defined as binary, it is not converted. If it is defined as string, it is converted as a string element (described below). If it is defined as a packed decimal value, it is converted as decimal with *Physical Type* set to `Packed Decimal` (described below).
- Decimal schema types: `decimal`, `integer`, `negativeInteger`, `nonNegativeInteger`, `nonPositiveInteger`, `positiveInteger` objects with *Physical Type* set to `External Decimal` are converted to the target `CodedCharSetId`. Elements with other *Physical Type* settings are converted to the target `Encoding`.
- Float schema types: `double`, `float` objects with *Physical Type* set to `External Decimal` are converted to the target `CodedCharSetId`. Elements with other *Physical Type* settings are converted to the target `Encoding`.
- Integer schema types: `byte`, `int`, `long`, `short`, `unsignedByte`, `unsignedInt`, `unsignedLong`, `unsignedShort` objects with *Physical Type* set to `External Decimal` are converted to the target `CodedCharSetId`. Elements with other *Physical Type* settings are converted to the target `Encoding`.
- String schema types: `anyURI`, `duration`, `ENTITIES`, `ENTITY`, `ID`, `IDREF`, `IDREFS`, `language`, `Name`, `NCName`, `NMTOKEN`, `NMTOKENS`, `normalizedString`, `NOTATION`, `QName`, `string`, `token` objects are converted to the target `CodedCharSetId` (the length of an object that has *Physical Type* of `Length Encoded String 2` is converted to the target `Encoding`).

### CWF Null handling options

The Custom Wire Format (CWF) supports handling of null values within messages. The Boolean Null Value that you set for the message set is applicable for all the defined objects within the message set.

For more information about the use of nulls, refer to the properties *Encoding Null* and *Encoding Null Value* for objects of each simple type, for example, “CWF properties for element reference and local element `dateTime` types” on page 289.

## Additional XML information

This section provides additional information in relation to the XML physical format. This information has been categorized into:

- “XML Null handling options”
- “XML rendering options” on page 678

### XML Null handling options

The XML Wire Format supports handling of null values within messages. *Encoding null* properties for XML are set only on the message set and apply to all the defined objects within the message set.

You can use the following two properties to represent the numeric and non-numeric encoding for NULL within the XML Wire Format:

- *Encoding Numeric Null*
- *Encoding Non-Numeric Null*

These represent the numeric and non-numeric encoding for NULL respectively.

- The numeric data types are:
  - Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger
  - Float schema types: double, float
  - Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedLong, unsignedShort
- The non numeric data types are:
  - Binary schema types: base64Binary, hexBinary
  - Boolean schema types: boolean
  - DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time
  - String schema types: anyURI, duration, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Each of these encodings has six enumerated values:

- NULLEmpty (default)
- NULLValue
- NULLXMLSchema
- NULLValueAttribute
- NULLAttribute (deprecated)
- NULLElement (deprecated)

You do not have to supply additional information for NULLEmpty, NULLXMLSchema and NULLValueAttribute, but if you select NULLValue, NULLAttribute, or NULLElement, you must define further values to be assigned to represent the NULL condition in the *Encoding Numeric Null Value* and *Encoding Non-Numeric Null Value* message set properties.

The following table shows how each encoding works. For each encoding, the example XML will cause the element myElem to be given a value NULL:

Encoding Numeric Null Encoding Non-Numeric Null	Encoding Numeric Null Value Encoding Non-Numeric Null Value	Example XML
NULLEmpty		<myElem/> <myElem></myElem>

Encoding Numeric Null Encoding Non-Numeric Null	Encoding Numeric Null Value Encoding Non-Numeric Null Value	Example XML
NULLValue	zzz	<myElem>zzz</myElem>
NULLXMLSchema		<myElem xsi:nil='true' /> <sup>1</sup>
NULLValueAttribute		<myElem></myElem> <sup>2</sup> <parent id="myElem"></parent> <sup>3</sup>
NULLElement	null <sup>4</sup>	<myElem><null /></myElem>
NULLAttribute	null <sup>4</sup>	<myElem null='true' /> <sup>1</sup>
<b>Notes:</b>		
<ol style="list-style-type: none"> <li>1. The attribute must evaluate to true, so the value must be either "true", "1" or the <i>Boolean True Value</i> property.</li> <li>2. This is only valid for <i>XMLElementAttrVal</i> element rendering, as specified in "XML rendering options" on page 678. Marking an element as being rendered in this way, and setting it to null, is equivalent to removing the attribute of the element that provides the element's value.</li> <li>3. This is only valid for <i>XMLElementAttrIDVal</i> element rendering, as specified in "XML rendering options" on page 678. Marking an element as being rendered in this way, and setting it to null, is equivalent to removing the attribute of the element that provides the element's value, but not removing the attribute that provides the element's name.</li> <li>4. Both <i>NULLElement</i> and <i>NULLAttribute</i> are deprecated. The element or attribute name provided should not include a namespace URI or prefix. If namespaces are enabled for the message set then the name matches any namespace.</li> </ol>		

You do not have to supply additional clarification for *NULLEmpty* and *NULLValAttr*, but if you select *NULLValue*, *NULLAttribute*, or *NULLElement*, you must define further values to be assigned to represent the NULL condition in the *Encoding Null Num Value* and *Encoding Null Non-Num Value* message set properties (see the table above).

#### XML Null value:

Unlike the TDS and CWF format, when you set the *Encoding Null Num* property to *NULLValue* in XML, the value is taken as a literal. A direct comparison is done with the text string, and no logical data conversion is performed.

For example, if you set the message set property *Encoding Null Num* to the value *NULLValue*, and you set *Encoding Null Num Val* to 0, a *FLOAT* value of 0.0 or a *DECIMAL* value of +0 does not match NULL.

If you set *Encoding Null Num* to *NULLEmpty*, this is equivalent to setting *Encoding Null Num* to *NULLValue* and *Encoding Null Num Val* to "".

#### XML Null element and NullValAttr:

In XML there are two conventions for storing a value:

1. It can be stored as an XML attribute with a local element or element reference property *Render* set to *XMLAttribute*, *XMLElement*, *XMLElementAttrID*, *XMLElementAttrVal*, or *XMLElementAttrIDVal*. For example, <element1 val="12"></element1>.
2. It can be stored as XML content with a local element or element reference property *Render* set to *XMLElement*. For example, <element1>12</element1>.

If you set the message set property *Encoding Null Num* to `NULLElement`, there is no way to represent a null value for an attribute value. If a null value is present in the tree (from ESQL or another format), an attribute with an empty string is written in the output message.

Conversely, if you have set the message set property *Encoding Null Num* or *Encoding Null Non-Num* to `NULLValAttr`, there is no way to represent a null value for a value rendered as XML content. If a null value is present in the tree, when writing an empty string, an element with no character content is written out instead.

### XML Null representation for Binary data:

If you use the *Encoding Null Non-Num Val* field in conjunction with a binary object in XML, you need to type the desired hex value. Do not insert the word `CDATA` in this field. If `CDATAHex` is specified in the *Encoding XML* property, `CDATA` rendering is used when writing the message.

### XML rendering options

There are four properties on the XML layer that you can use to affect how the XML messages are rendered. The table below shows examples of the values that you can set for the *Member Render* property. In this table, the member element is referred to as *A*, and has the value `value of element`. The parent is referred to as *X*.

### The effect of rendering options on XML output

To get XML rendered like this:	Set this Member Render property value:	Set these other property values:
<X> <A>value of element</A> </X>	XMLElement (the default)	Member XML Name = A
<X A='value of element' />	XMLAttribute	Member XML Name = A
<X> <Field id='A'>value of element</Field> </X>	XMLElementAttrID	Member XML Name = Field Member ID Attribute Name = id Member ID Attribute Value = A
<X> <A val=value of element /> </X>	XMLElementAttrVal	Member XML Name = A Member Value Attribute Name = val
<X> <Field id='A' val='value of element' /> </X>	XMLElementAttrIDVal	Member XML Name = Field Member ID Attribute Name = id Member ID Attribute Value = A Member Value Attribute Name = val

You are not recommended to have an element in the model that is rendered as an XML attribute. This can result in incorrect validation of XML documents. Instead the element should be redefined as an attribute in the model.

You are not recommended to have an attribute in the model that is rendered as an XML element. This can result in incorrect validation of XML documents. Instead the attribute should be defined as an element in the model.

There is one scenario where this technique is appropriate. When you have created a message model by importing a C header file or a COBOL copybook, it will consist entirely of elements. An XML form of this model can be created by simply adding an XML physical format to the message set. If you are looking for certain elements to appear as XML attributes in the XML form, then you can use the Render property to achieve this.

## Additional TDS information

This section provides additional information in relation to the TDS physical format. It has been categorized into:

- “TDS Industry standard formats”
- “Message characteristics” on page 684
- “TDS Null handling options” on page 691
- “TDS message model integrity” on page 692
- “Using regular expressions to parse data elements” on page 695

### TDS Industry standard formats

WebSphere Message Broker supports the ACORD AL3 , EDIFACT, FIX, HL7, SWIFT, TLOG, and X12 standards. For some of these standards, default property values are supplied as defined in “Default TDS Message set properties” on page 169. If you use these defaults, or override some of these defaults where necessary, you can model all these industry standard formats.

For more details about each of these industry standards see:

- “EDIFACT messaging standard”
- “HL7 messaging standard” on page 680
- “SWIFT messaging standard” on page 681
- “TLOG messaging standard” on page 682
- “X12 messaging standard” on page 682
- “ACORD AL3 messaging standard” on page 682
- “FIX messaging standard” on page 684

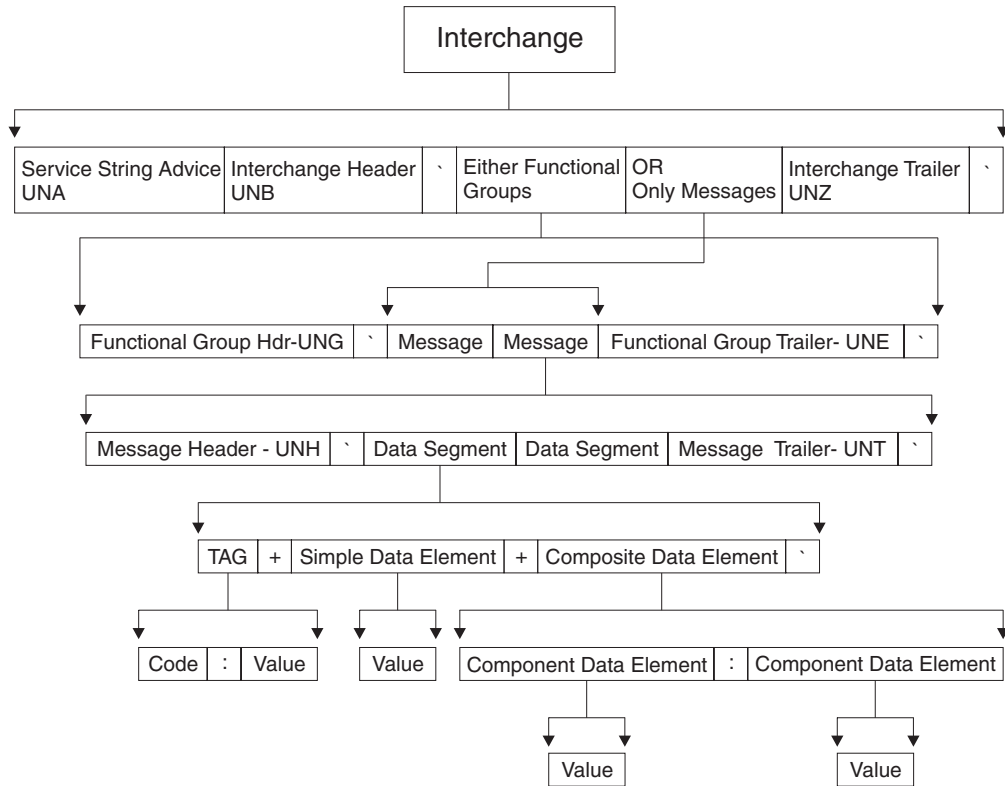
These topics also contain details of any pre-defined message set solutions available from IBM.

#### **EDIFACT messaging standard:**

EDIFACT is an international standard for EDI trading in commercial and non-commercial sectors. EDIFACT has an underlying syntax, which is an ISO standard. Within that syntax, there are directories of data elements, composite data elements, segments, and messages. There are conventions for placing messages in an “envelope” which identifies the sender and receiver and other attributes of a transmission. For more information on the EDIFACT messaging standard, see the United Nations Centre for Trade Facilitation and Electronic Business Web site and click “Standards” on the left side.

EDIFACT messages can be modeled using the MRM Tagged/Delimited String Format (TDS)

The high level structure of an EDIFACT message is shown below.



You can model the top level interchange of an EDIFACT message by setting the following properties for the complex type on which the message is based:

*Composition* = Sequence  
*Content Validation* = Closed  
*Tag Data Separator* = <EDIFACT\_TAGDATA\_SEP>  
*Data Element Separation* = Tagged Delimited  
*Delimiter* = <EDIFACT\_CS>

Within an EDIFACT message, you can define the delimiters to be used in the message itself using the optional Service String Advice element. To enable this element to be recognized as an EDIFACT Service String, you must set the element property *Interpret Element Value* to EDIFACT Service String. You must also set the delimiter values to the mnemonic values that are defaulted when you set the *Message Standard* property to EDIFACT.

A pre-defined message set solution for EDIFACT may be purchased from IBM.

### HL7 messaging standard:

Hospitals, doctors, healthcare professionals and institutions require the ability to exchange information. The HL7 messaging standard defines the structure and content of the messages that are exchanged between systems that are working in various administrative, financial, and clinical activities in the healthcare industry.

HL7 messages can be modeled using the MRM Tagged/Delimited String Format (TDS).



If you are working with HL7 messages you can specify the messaging standard at the message set level and a number of the properties for this standard are set to default settings for HL7 at the message set, complex type, group and element levels.

A pre-defined message set solution for HL7 may be purchased from IBM.

**SWIFT messaging standard:**

SWIFT is a cooperative owned by the financial industry. It supplies secure, standardized messaging services and interface software to financial institutions. For more information on the SWIFT messaging standard, see the SWIFT community Web site.

SWIFT FIN messages can be modeled using the MRM Tagged/Delimited String Format (TDS).

The high-level block structure of a SWIFT message is shown in the table below.

SWIFT message high level block structure

Block name	Format
Basic header	{1:...}
Application header	{2:...}
User header	{3:...}
Text	{4:...}
Trailer	{5:...}

When they are concatenated in a message, the blocks appear as:

{1:...}{2:...}{3:...}{4:...}{5:...}

You can model this setting the following type properties for the message:

```
Data Element Separation = Tagged Delimited
Group Indicator = {
Delimiter = }{
Group Terminator = }
Tag Data Separator = :
```

Each block is modeled as a complex element with element *Tag* property values of 1,2,3,4, and 5 respectively.

The text body of the message has the following format:

```
{4:
:20:X
:32A:940930USD1,
.....
:72:/A/
-}
```

You can model the complex type of the Text body by setting the following type properties:

*Data Element Separation* = Tagged Delimited  
*Group Indicator* = <CR><LF>:  
*Delimiter* = <CR><LF>:  
*Group Terminator* = <CR><LF>-  
*Tag Data Separator* = :

The *Tag* property of the elements within the body has values of 20, 32A, 72, and so on.

A pre-defined message set solution for SWIFT may be purchased from IBM. See the WebSphere MQ SupportPacs Web page.

#### **TLOG messaging standard:**

In the retail industry, a TLOG is the Point of Sale (POS) Transaction Log. The TLOG is a complete, detailed record of everything that occurs at the POS terminal - including events that are not directly related to a sales transaction. Typically, the precise TLOG record format is unique to a given POS application, but the majority of formats are based on a tagged/delimited string format called Raw TLOG.

Raw TLOG messages can be modeled using the MRM Tagged/Delimited String Format (TDS).

If you are working with TLOG messages you can specify whether fields in the messages are character format or a format specific to the message. This requires the *Messaging Standard* property (at the message set level) to be set to TLOG and relevant objects that have this non-character based field in the TDS message to have their *Physical Type* property set to Messaging Standard Alternate.

Pre-defined message set solutions for TLOG may be purchased from IBM.

#### **X12 messaging standard:**

X12 is a standard for EDI trading in commercial and non-commercial sectors. X12 has an underlying syntax, which is an ANSI standard. Within that syntax, there are directories of data elements, composite data elements, segments, and messages. There are conventions for placing messages in an "envelope" which identifies the sender and receiver and other attributes of a transmission. For more information on the X12 messaging standard, see the ASC X12 Web site.

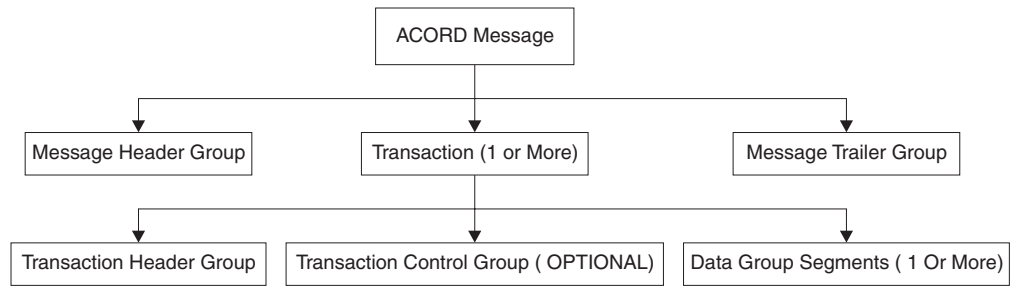
X12 messages can be modeled using the MRM Tagged/Delimited String Format (TDS).

If you are working with X12 messages, you can define the delimiters to be used in the message itself using the mandatory Interchange Control Header element. To enable this element to be recognized as an X12 Service String, you must set the element property *Interpret Element Value* to X12 Service String. You must also set the delimiter values to the mnemonic values defaulted by setting the *Message Standard* property to X12.

A pre-defined message set solution for X12 may be purchased from IBM.

#### **ACORD AL3 messaging standard:**

The basic structure of an ACORD AL3 message is shown below.



Each group with an ACORD AL3 message has a header consisting of a one-digit number, three letters, plus a three-digit total length count. These first seven characters can be modeled as a tag. The data within the headers is fixed length. Therefore the header type used for the overall message can be modeled as follows:

*Data Element Separation* = Tagged Fixed Length  
*Length of Tag* = 7

The Transaction Group contains other groups, and is therefore modeled in the same way as the overall message. The Message Header Group and the Message Trailer group just consist of fixed length elements, therefore the type used can be modeled as:

*Data Element Separation* = Fixed Length

There are two *Data Element Separation* methods particularly suited to handling ACORD AL3 messages:

- Fixed Length AL3 supports basic handling of ACORD AL3 messages, including situations where the message groups conform to a different version of the ACORD AL3 standard. This is deprecated and will be removed in a future version of the product and an alternative will be provided.
- Tagged Encoded Length supports handling of more sophisticated situations, including messages containing message groups unknown to the message dictionary.

The following sections describe their use:

- “Using Fixed Length AL3”
- “Using Tagged Encoded Length to support re-versioning” on page 684

*Using Fixed Length AL3:*

This is deprecated and will be removed in a future version of the product and an alternative will be provided.

You can select the value Fixed Length AL3 for the *Data Element Separation* property for complex types within a message that conforms to the ACORD AL3 standard. This allows different versions of the ACORD AL3 standard to be supported using the same message set. This value is similar to the value Fixed Length except for the following:

- A question mark (?) in the left-most position of an element means that it is skipped.
- A sequence of question marks is inserted for all missing optional elements.
- Unused trailing optional elements are truncated.
- Any <CR><LF> after the last element is ignored.

- The length field is extracted on input (and *not* put to the tree), and automatically recalculated on output.

*Using Tagged Encoded Length to support re-versioning:*

The incoming message might contain a group that is no longer in use within the current ACORD AL3 standards, and has therefore been deleted from the later version of the standards. Similarly, the incoming bit stream might be from a later version of the ACORD AL3 standards, and might contain a new group that was not defined in earlier versions.

In order to correctly parse this self defining tag, the TDS parser needs to know the length of the group it is parsing and skip to the end of all data associated with that self defining tag.

Use the *Data Element Separation* method Tagged Encoded Length to handle these situations. You will also need to set these properties:

- Length of Tag or Tag Data Separator, so that the TDS parser knows where tags end.
- Length of Encoded Length, so that the TDS parser knows the size of the length field.
- Extra Chars in Encoded Length, are used to indicate to the TDS parser how many characters, apart from the data itself, are counted in the encoded length field.

#### **FIX messaging standard:**

The Financial Information eXchange (FIX) Protocol is a series of messaging specifications. It is a global language describing trade-related messages, and is used for automated trading of securities, derivative, and other financial instruments. For more information on the FIX protocol, see the FIX protocol Web site.

FIX messages can be modeled using the MRM Tagged/Delimited String Format (TDS).

A pre-defined message set solution for FIX may be purchased from IBM. See the WebSphere MQ SupportPacs Web page.

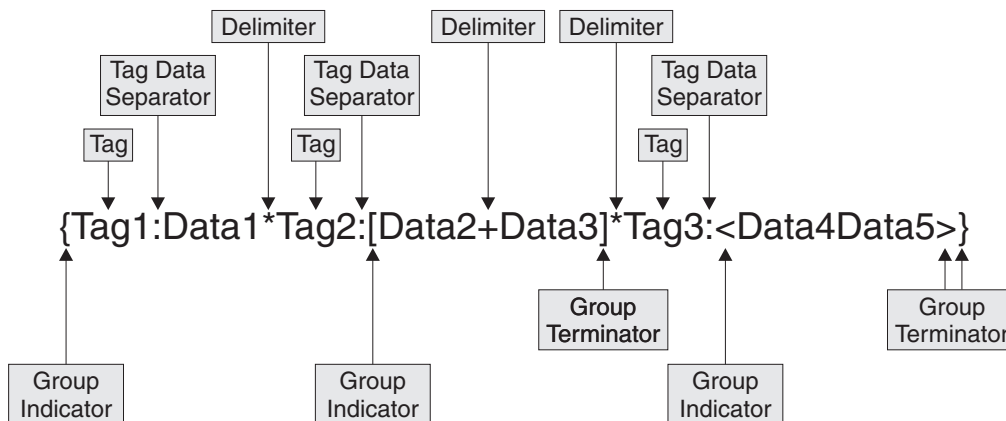
#### **Message characteristics**

There are a number of features of text string messages that are common across many formats. The following sections give an overview of the main features that are supported by the TDS wire format:

- The text strings in the message can have a tag or a label preceding the data value. The tag is a string that uniquely identifies the data value. The TDS format allows you to associate a tag with each element when you define the element in the workbench.
- The message can contain various special characters or strings in addition to the tags and text string data values. The TDS format supports a number of different types of special characters or strings. Some messages have a special character or string that separates each data value from the next. In the TDS format this is a known as a delimiter. In formats that have a tag before each data value, the tag can be separated from its data value by a special character or string. In the TDS format this is known as a tag data separator.

- A message can be split into a number of substructures in a similar manner to a COBOL or C structure. You can model each of these substructures separately by defining complex types or elements for each one. Complex types and elements are described in “Message model objects” on page 16. A substructure can have a special character or string that indicates its start within the data. This is known in the TDS format as a group indicator. A substructure can also have a special character or string that indicates its end in the data. In the TDS format, this is known as a group terminator. A group indicator and group terminator can also be defined for the whole message. Group indicators and group terminators are optional for the message and each substructure.
- Some text strings within a message can be of fixed length, so a delimiter between each data value is not necessary. This is supported by the TDS format. If you use a fixed length tag, a tagged data separator is not required.
- The TDS property that controls the way text strings are separated is *Data Element Separation*. It has several options that let you choose, for example, if tags are used, if strings lengths are fixed or variable, and what types of text strings are permitted. See “Specifying data element separation methods to model a message” on page 686.
- The substructures within a message can use different types of *Data Element Separation* and use different special characters. Therefore the TDS format allows you to define different types of data element separation and special characters for each complex type within the message.
- If you use the Use Data Pattern method of *Data Element Separation*, you can use regular expressions to identify parts of the message data to be assigned to sub-fields. This is done by setting the regular expression in the Data Pattern property. See “Using regular expressions to parse data elements” on page 695 for further details.

The figure below illustrates the tags and special characters in a TDS message, showing an example data message with each of its components labeled.



- At the top level, each data value has a tag associated with it, each tag is separated from its data value using a tag data separator of colon (:), and the data values are separated from each other using the asterisk delimiter (\*).
- The group indicator for the message is the left brace ( { ) and the group terminator is the right brace ( } ).

- The data values Data2 and Data3 are in a substructure in which there are no tags, and each data element is separated from the next using the delimiter plus (+). The group indicator for this substructure is the left bracket ([]) and the group terminator is the right bracket (]).
- The data values Data4 and Data5 are in a substructure in which the values are fixed length, and are therefore not separated by a delimiter. The group indicator for this substructure is the less than symbol (<) and the group terminator is the greater than symbol (>).

The following sections describe data element separation and the special characters in more detail:

- “Specifying data element separation methods to model a message”
- “Specifying special characters to model a message” on page 688

### Specifying data element separation methods to model a message:

Elements of data in a TDS message are identified according to the data element separation method that you must specify for the *Data Element Separation* property for a complex type. Depending on the value that you have set for *Data Element Separation*, the properties *Tag Data Separator* and *Delimiter* (for a message set and a complex type) might also be required to identify each element.

The methods that you can specify for each complex type are described below. The examples given are all based on a complex type that contains three elements of type STRING. The *Tag Data Separator*, where used, is the colon (:), and the *Delimiter*, where used, is the asterisk (\*).

#### Tagged Delimited

Each data value is preceded by a tag that is specified as an element property. If the tag has an associated *Length of Tag*, indicating that the tag has a fixed length, each data value follows immediately after the tag. If the tag is not specified as fixed length, the tag is separated from the next element by a *Tag Data Separator*. Each data value is separated from the next by a *Delimiter*. There is no *Delimiter* after the last element in the complex type.

The following example shows tags of fixed length:

```
tag1data1*tag2data2*tag3data3
```

The following example shows tags of variable length:

```
tag1:data1*tag11:data2*tag111:data3
```

#### Tagged Fixed Length

This method is similar to Tagged Delimited, but the data values are always fixed length. Therefore, no delimiter is required after each data value. The tags themselves can be fixed length or variable length, depending on the setting of *Tag Data Separator* and *Length of Tag*.

The following example shows tags of fixed length:

```
tag1data1tag2data2tag3data3
```

The following example shows tags of variable length:

```
tag1:data1tag11:data2tag111:data3
```

#### Tagged Encoded Length

This method has a tag and a length field before the data. It indicates to the parser that following each tag in the bit stream there is data defining the

length of data to be associated with that tag. You must set the *Length of Encoded Length* parameter. If the value in *Length of Encoded Length* includes extra characters, you must also set the *Extra Chars in Encoded Length* parameter.

The following example shows a tag of fixed length of four characters (*Length of Tag* has been set to four), a three-character length field (*Length of Encoded Length* has been set to three), and several characters of data. *Extra Chars in Encoded Length* has been set to zero:

```
tagA007dataAAAtagB006dataBBtagC009dataCCCC
```

Given the bit stream above, the parser finds the tag "tagA" and extracts the length value 7. Because *Extra Chars in Encoded Length* is set to zero, the next seven (7 - 0) characters are the data. Then follow the characters for the next tag "tagB" and the length value of 6, and so on for tag "tagC". In each case in this example, the value in the length field is exactly the length of data.

The following example shows tags with a fixed length of four characters (*Length of Tag* has been set to four), a three-character length field (*Length of Encoded Length* has been set to three), and several characters of data. *Extra Chars in Encoded Length* has been set to three (because in this example the length field value includes the three-character length field as well as the data field):

```
tagA012dataAAAAAtagB010dataBBBtagC016dataCCCCCCCC
```

Given the bit stream above, after "tagA" the parser extracts the length value 12. But because *Extra Chars in Encoded Length* is set to three, only the next nine (12 - 3) characters are the data. Then follow the characters for "tagB" and length value 10, and so on. In each case in this example, the value in the length field is three more than the actual length of data.

### All Elements Delimited

The data values have no tag, but each data value is separated from the next by a delimiter.

The following example shows this:

```
data1*data2*data3
```

### Variable Length Elements Delimited

If a data element is fixed length, the next data value follows immediately after it. If the data element is variable length, the next data value is separated from it by the delimiter. There are no tags.

The following example shows element 2 as fixed length, and elements 1 and 3 as variable length:

```
data1*data2data3
```

### Use Data Pattern

The data associated with each element is determined by the parser matching the data with the regular expression in the *Data Pattern* property for that element. The TDS parser uses the regular expression in the *Data Pattern* to:

- Determine the length of data to associate with each element.
- Determine if, in the case of a repeating element, another occurrence of an element is present in the bit stream.
- Determine the presence (if the pattern is matched) or absence (if the pattern is not matched) of an element in the bit stream.

There are no delimiters or tags, other than those coded as part of the regular expression patterns. See “Regular expression syntax” on page 697 for an explanation of how pattern matching works.

The following example shows three elements, each having the regular expression *Data Pattern* shown:

```
First Data Pattern = [A-Z]{1,3}
Second Data Pattern = [0-9]+
Third Data Pattern = [a-z]*
```

```
Message data = 'DT31758934information for you'
```

```
First element data: 'DT'
Second element data: '31758934'
Third element data: 'information'
```

The first *Data Pattern* means “from one to three characters in the range A to Z”, the second means “one or more characters in the range 0 to 9”, and the third means “zero or more characters in the range a to z”. Notice how each element’s data was terminated by the first character that did not match the element’s *Data Pattern*.

### Fixed Length

All elements are fixed length, and each data value immediately follows the next with no delimiter. There are no tags.

The following example shows this:

```
data1data2data3
```

### Fixed Length AL3

This method is the same as Fixed Length, but it also notifies the parser to implement a number of rules in relation to missing elements, length encoding, and versioning that are predefined in the ACORD AL3 standard.

### Undefined

This value is set automatically when you set the *Type Composition* property of a complex type to Message, and you cannot set it to any other value. You are also unable to set values for the TDS Type properties *Group Indicator*, *Group Terminator*, *Tag Data Separator*, *Length of Tag*, and *Delimiter*.

If you set the *Data Element Separation* method to Undefined, you must not set the *Type Composition* property to Empty, Choice, Unordered Set, Ordered Set, Sequence, or Simple Unordered Set.

For more information about *Type Composition* set to Message, see “Multipart messages” on page 28.

### Specifying special characters to model a message:

You can specify a number of different types of special character in the workbench. You can specify special character values for message sets, types, and type members. The values that you set for a type override the corresponding values set for the message set in which it is defined.

You can specify a special character value in one of the following ways:

1. As a literal string of one or more characters.
2. As a mnemonic value.
3. As a combination of both mnemonics and literals.



The types of special character are described in the table below.

Special character type	Description	Set as a property of...
Group Indicator	This is a string that indicates the start of a group or complex type within a message	Message set, type
Group Terminator	This is a string that indicates that the end of a group or complex type within a message	Message set, type
Tag Data Separator	This is the string that is used to separate a tag from its data.	Message set, type
Delimiter	This is the string used to separate data elements from one another	Message set, type
Repeating Element Delimiter	This is the string used to separate repeating data elements from one another	Type member

Therefore, if you create a complex type and set *Data Element Separation* property to Tagged Delimited, the *Group Indicator* property to left brace ({}), the *Group Terminator* to right brace (}), the *Tag Data Separator* to colon (:), and the *Delimiter* to asterisk (\*), the bit stream has the following format:

```
{tag1:data1*tag2:data2*tag3:data3}
```

In some message formats, a special character is specified before each element or after each element, as shown in the following two examples:

```
:data1:data2:data3
```

```
data1:data2:data3:
```

You can model these formats by using a combination of the *Data Element Separation* method, the *Delimiter* value, the *Group Indicator* value, and the *Group Terminator* value.

For the first example, specify *Data Element Separation* as All Elements Delimited, *Delimiter* as colon (:), and *Group Indicator* as colon (:).

For the second example, specify *Data Element Separation* as All Elements Delimited, *Delimiter* as colon (:), and *Group Terminator* as colon (:).

*Using mnemonics as special characters:*

A mnemonic is a tag delimited by < and >. The broker translates the mnemonic to obtain the actual value of the special character. There are two types of mnemonic:

- Control code mnemonics which map to the common non-printing characters. These are mapped using the local codepage for your system. This is typically an ASCII codepage on distributed platforms and an EBCDIC codepage on other platforms. This means that characters are generally mapped to the 'expected' values for your system. This does depend on your codepage setting and if in doubt you should refer to your system documentation. In the event that a specific mnemonic is not mapped to the value you need then you can use the explicit <U+xxxx> representation described below.
- Message mnemonics for use with specific industry message standards such as X12.

These are mapped according to their associated message standard. Each mnemonic has a default mapping, but in message standards such as EDIFACT and X12 this default can be overridden by a 'service string' specified in the message itself.

Mnemonics can be specified in one of two ways:

1. <Mnemonic\_Name> where Mnemonic\_Name can comprise alphanumeric characters and underscore (\_) characters.
2. <U+xxxx> where xxxx are hexadecimal numbers. The delimiter is interpreted as the Unicode character that corresponds to the hexadecimal value.

In addition to special characters, mnemonics can also be used in the message set properties Decimal Point, Escape Character, and Reserved Characters.

For more details about the supported mnemonics, see "TDS Mnemonics" on page 168.

*TDS Mnemonics:*

The Tagged/Delimited String Format (TDS) uses mnemonics for a number of properties for a message set, complex type, or both. These TDS mnemonics and their associated properties are listed in the table below.

Mnemonic string	Meaning	Default value	Associated property
<EDIFACT_CS>	Component separator in EDIFACT	:	Message set and complex type/group, <i>Delimiter</i>
<EDIFACT_DS>	Data element separator in EDIFACT	+	Message set and complex type/group, <i>Delimiter</i>
<EDIFACT_TAGDATA_SEP>	Tag data separator in EDIFACT  This is overridden with the same value as that which overrides <EDIFACT_DS>	+	Message set and complex type/group, <i>Tag Data Separator</i>
<EDIFACT_DEC_NOTATION>	Decimal notation in EDIFACT	.	Message set, <i>Decimal Point</i>
<EDIFACT_ESC_CHAR>	Escape character in EDIFACT	?	Message set, <i>Escape Character</i>
<EDIFACT_GROUP_TERM>	Tag terminator in EDIFACT	'	Message set, <i>Group Terminator</i>
<X12_GROUP_TERM>	Tag terminator in X12	!	Message set level, <i>Group Terminator</i>
<X12_DS>	Data element separator for X12	*	Message set and complex type/group, <i>Delimiter</i>
<X12_CS>	Component separator for X12	:	Message set and complex type/group, <i>Delimiter</i>
<HL7_CS>	Component separator in HL7	^	Message set and complex type/group, <i>Delimiter</i>
<HL7_FS>	Data element separator in HL7		Message set and complex type/group, <i>Delimiter</i>
<HL7_RS>	Repeating element delimiter in HL7	~	Local element and element reference, <i>Repeating Element Delimiter</i>
<HL7_SCS>	Sub-component separator in HL7	&	Message set and complex type/group, <i>Delimiter</i>

Mnemonics are also supported for the following control characters:

<ACK> (x'06')	<BEL> (x'07')	<BS> (x'08')	<CAN> (x'18')
<CR> (x'0D')	<DC1> (x'11')	<DC2> (x'12')	<DC3> (x'13')
<DC4> (x'14')	<DLE> (x'10')	<EM> (x'19')	<ENQ> (x'05')
<EOT> (x'04')	<ESC> (x'1B')	<ETB> (x'17')	<ETX> (x'03')
<FF> (x'0C')	<FS> (x'1C')	<GS> (x'1D')	<GT> (x'3E')
<HT> (x'09')	<LF> (x'0A')	<LT> (x'3C')	<NAK> (x'15')
<NUL> (x'00')	<RS> (x'1E')	<SI> (x'0F')	<SO> (x'0E')
<SOH> (x'01')	<SP> (x'20')	<STX> (x'02')	<SUB> (x'1A')
<SYN> (x'16')	<US> (x'1F')	<VT> (x'0B')	

You can enter a mnemonic in the form <U+xxxx> where xxxx are hexadecimal numbers up to a value of FFFF. These numbers represent a Unicode character, not a character in your local code page or the code page in which message data is formatted. None of the characters in this structure are case sensitive. Do not enclose spaces inside the angle brackets.

## TDS Null handling options

TDS supports handling of null values within messages. You can use the message set property *Boolean Null Representation* to specify the value to be used for Boolean Null representation. You can use the object properties *Encoding Null* and *Encoding Null Value* to control how null handling is represented for individual objects.

You can select the *Encoding Null* property from the three enumerated values `NULLPadFill`, `NULLLogicalValue`, and `NULLLiteralValue`:

- You should use the `NULLPadFill` option only for fixed length objects. If you select this option for an object of simple type `dateTime`, a null `dateTime` is written out, which is an empty tag with a delimiter. (This is equivalent to selecting `NULLLiteralValue`, with the *Encoding Null Value* property set to the empty string `""`.) If you select this option for an object of another simple type, the object is filled with the value specified by the *Padding Character* property. If you select this option, the *Encoding Null Value* property is disabled.

If you use this option for a variable length object, the parser does not know how many padding characters to write out, so it does not write any. Instead, the parser writes an explicit null, with tag and delimiter but no data value. For example:

```
tag1:,
```

is written out, where tag1 is the tag for the variable length element with `NULLPadFill` set, ":" is the tag data separator, and "," is the delimiter.

- If you select the `NULLLogicalValue` option, the value entered for the *Encoding Null Value* property is converted to its logical value. For writing, the logical value is written in the same way as any other value. For parsing, the converted logical value is compared against the converted message data.
- If you select the `NULLLiteralValue` option, the value entered for the *Encoding Null Value* property is directly substituted as if it were a string value. The value is case insensitive. For fixed length objects, the literal value must be no longer than the length of the object.

If the literal value is shorter, the *Encoding Null Value* is padded (using *Padding Character*) on output. On input, if the `NULLLiteralValue`'s length does not match the *Length* field, you should set the message set level *Trim Fix Len String* property so that padded nulls are correctly parsed.

The use of the *Encoding Null Value* property is dependent on the value that you select for the *Encoding Null* property described above. Null values are not defined for binary types. The properties *Encoding Null* and *Encoding Null Value* are therefore not set for binary types.

## TDS message model integrity

When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. These rules are checked any time the project is saved. If an inconsistency is found, the error is displayed in the task list of the workbench.

The following sections cover the rules for TDS wire format properties:

- “General rules: TDS message model integrity”
- “Restrictions for nesting complex types” on page 694
- “Omission and truncation of elements” on page 694

### General rules: TDS message model integrity:

This section describes the general rules for each value that you can set for the *Data Element Separation* property of a type.

#### Tagged Delimited

- The *Tag* property for every simple child element must contain a non-empty value.

#### Tagged Encoded Length

- The *Tag* property for every simple child element must contain a non-empty value.
- The *Length Of Encoded Length* property must contain a positive integer greater than zero.

#### Variable Elements Delimited

- The *Delimiter* property must contain a non-empty value.

#### Use Data Pattern

- Each simple element that is a child of the complex type must have a regular expression specified for *Data Pattern*. See “Regular expression syntax” on page 697.

#### All Elements Delimited

- The *Delimiter* property must contain a non-empty value.

#### Fixed Length

- All simple child elements must specify a length, unless their data type is boolean (or derived from boolean).
- All compound child elements must specify a length, unless their data type is boolean (or derived from boolean).
- The length can be specified using either the *Length* property, or the *Length Value Of member* property.

#### Fixed Length AL3

- All complex child elements with a non-boolean compound element and non-boolean simple child elements must have either a nonzero value in their *Length* property, or a non-empty value for their *Length Value Of type member* property.

## Tagged Fixed Length

- All complex child elements with a non-boolean compound element and non-boolean simple child elements must have either a nonzero value in their *Length* property or a non-empty value for their *Length Value Of* type member property.
- The *Tag* property for each and every simple child element must contain a non-empty value.

The following rules also apply:

- If you have set the parent *Type Composition* to Choice, and the parent *Data Element Separation* property to Variable Elements Delimited, All Elements Delimited, Fixed Length, or Fixed Length AL3:
  - You must not set the *Type Composition* to Message for any child complex types.
  - You must not set the *Data Element Separation* method to Tagged Delimited or Tagged Fixed Length for any child complex types.

If you do so, the message set will not deploy successfully.

- If you have set the type's *Data Element Separation* property to Fixed Length, Fixed Length AL3, or Tagged Fixed Length, you must set either the *Length* or *Length Value Of* property for all simple elements under this parent, and also for all complex elements with a simple content and compound elements.
- For a Choice in a fixed length environment (*Data Element Separation* set to Fixed Length, Tagged Fixed Length, or Fixed Length AL3), length references are not valid, and element lengths should be used.
- Elements specified in a *Length Value Of* property must be simple elements of type INTEGER, they must exist in the same structure as the referring element, and they must appear before the referring element in that structure.
- Complex types with simple content and Compound elements must have an empty *Length Value Of* type member property. This is because the *Length Value Of* element would occur after the referring element in the parent structure, which is disallowed by the previous rule.
- Complex types with simple content cannot have a separation type of Use Data Pattern.
- Compound elements cannot have a separation type of Use Data Pattern.
- Regardless of the setting of the type's *Data Element Separation* property, if the type of a simple element is BINARY, you must set either the *Length* or *Length Value Of* property.
- For fixed length elements, the *Justification* property must be set to something other than Not Applicable, and the *Padding Character* property cannot be an empty value.
- If any element within a message has its *Interpret Element Value* property set to Message Key, the *Message Key* property must be set for all messages within the message set.
- If you have set the *Repeat* property in the type member to Yes, you must set a value for the *Max Occurs* property in the following two situations:
  - If you have defined an element as a member of a complex type that has the property *Data Element Separation* set to Fixed Length.
  - If you have defined a fixed length element as a member of a complex type that has the property *Data Element Separation* set to Variable Elements Delimited.

When it is invoked by the broker to interpret an input message, the parser assumes that the number of occurrences of the element is equal to the value that

you set for *Max Occurs*. When the parser constructs an output message, if there are fewer elements than the value of *Max Occurs*, the missing elements are inserted with default values.

### Restrictions for nesting complex types:

If you include a group within another group or complex type, the *Data Element Separation* property for the nested group must be compatible with the *Data Element Separation* property of the parent group or complex type. For example, you cannot set the parent property to Fixed Length and the child property to Tagged Delimited, because the length of the Tagged Delimited structure would not be known, and would therefore conflict with the parent definition. If groups are nested to three or more levels, the *Data Element Separation* property for each nested group must be compatible with all of its parent groups.

The rules for compatibility are listed in the table of permitted options for nested complex types shown below.

	Parent				
Child	Tagged Delimited, Tagged Encoded Length	All Elements Delimited, Variable Elements Delimited	Fixed Length, Fixed Length AL3	Tagged Fixed Length	Use Data Pattern
Tagged Delimited, Tagged Encoded Length	Allowed	Allowed	Not allowed	Not allowed	Allowed
All Elements Delimited, Variable Elements Delimited	Allowed	Allowed	Not allowed	Not allowed	Allowed
Fixed Length, Fixed Length AL3	Allowed	Allowed	Allowed	Allowed	Allowed
Tagged Fixed Length	Allowed	Allowed	Not allowed <sup>1</sup>	Allowed	Allowed
Use Data Pattern	Allowed	Allowed	Allowed	Allowed	Allowed

**Note:**

1. Tagged Fixed Length cannot exist at the inner level if any outer level has a *Data Element Separation* method of Fixed Length or Fixed Length AL3. This is because an item of Tagged Fixed Length can repeat a variable number of times. Fixed Length and Fixed Length AL3 are parsed by moving a set number of bytes: with a variable number of repeats, it is not possible to calculate the number of bytes that need to be parsed.

### Omission and truncation of elements:

The omission and truncation of elements is dependent on the setting of the property *Suppress Absent Element Delimiters*. A description of this can be found in “Complex type TDS properties” on page 219, “Global group TDS properties” on page 224, or “Local group TDS properties” on page 229.

If you have created a message in which some elements are optional, an input message might not contain all defined elements. If the elements are in a complex

type that you have defined with the *Data Element Separation* property of the type set to All Elements Delimited or Variable Elements Delimited (in which the elements are separated by a delimiter and have no tag), any elements that are missing from the end of the complex type must be indicated by the application that creates the message in one of two ways. These both provide techniques to avoid unnecessarily long sequences of delimiters, and to preserve consistent representation of missing elements.

1. If you have set the *Delimiter* property for the complex type to a value that does not match the value that you have set for the *Delimiter* property for any of the complex type's parent types, the elements at the end of the message can be indicated by the occurrence of a *Delimiter* of one of its parents after the last actual element in the complex type data.

This is known as the truncation method, in which missing elements are treated as not expected, and both data and delimiters are omitted in the bit stream.

For example, you define a complex element C that has four optional elements. You set the *Delimiter* property to the character plus (+). You define complex element P, and set the *Delimiter* property of P to asterisk (\*). You add three elements to P, the first is a string, the second is complex element C, and the third is a string.

When a particular instance of the message is received by the broker, all the elements of P are present, but only the first two elements of C are present. The data in the message appears as follows if the truncation method is used (where P<sub>n</sub> are the values of the elements of P and C<sub>n</sub> the values of the elements of C):

P1\*C1+C2\*P3

When the parser encounters the second asterisk delimiter, it determines that the last two elements of complex element C are not present, and the next element is the third element of P.

You can use truncation successfully only when both omission and truncation cause the parser to exhibit the same behavior, unless the elements truncated are fixed length.

2. If the *Delimiter* of the complex type matches that of one of its parents, the truncation method cannot be used. This is because the parser cannot determine whether the delimiter after the last element is for the current complex type, or for one of its parents. Therefore a delimiter must be included in the message data for each missing element to ensure that the parser can match the elements with the model.

This is known as the omission method, in which missing simple elements are represented by an empty sequence of characters between two delimiters.

For example, you define P and C as in the previous example, but set the *Delimiter* property for P to plus (+). When the same message is received by the broker (all elements of P are present, the first two elements of C are present), the data in the message appears as follows:

P1+C1+C2++P3

Two delimiter characters have been inserted in the message data for the missing elements of complex element C. If the truncation method had been used, the parser would have interpreted the data value P3 as the value of the third element of complex element C and not the third element of complex element P.

## Using regular expressions to parse data elements

If your input messages can contain optional sub-fields whose presence or absence can only be determined by examining the actual value of the data (for example an

optional field of numeric digits followed by one or more alphabetic characters) you need to use the *Data Element Separation* method Use Data Pattern. This is particularly relevant to messages conforming to the SWIFT industry standard. To use this method, you must provide regular expressions to identify those portions of an input message that are to be associated with sub-fields. You need to provide a regular expression value for the *Data Pattern* property of each child of the complex type.

When parsing, data is matched in turn to each child of the complex type. The parser does this by using the regular expression for the child to determine the number of characters from the message that apply for that child. This number of characters is the length of the longest string, starting from the current position in the message, that matches the regular expression. If the longest string that matches the regular expression is of length zero, the element is present in the message, and the empty string is used for the value. If no string matches the regular expression, the element is not present. This might cause a subsequent validation error if the element is required.

After the number of characters from the input message has been determined, normal data conversion, or further parsing in the case of a complex element, is performed on the text of the input message to assign values to elements. This might lead to data overrun or underrun errors if the length identified by the pattern is not appropriate for the definition of the child.

“Regular expression syntax” on page 697 explains the full syntax rules and how to apply them, but the table below gives a few simple examples of parsing using data patterns. A more complex example appears after the table.

Input message	Data Pattern	Value matched
"123456ABC"	[0-9]*	"123456"
"123"	[A-Z]*	""
"123"	[A-Z]+	Not present
"ABCD123"	[A-Z]{1,3} first field [A-Z]{2,4} second field	"ABC" - first field (the longest string matching the pattern) Not present - second field (minimum length of two alphabetic characters is not present)
"ABCDEFGHJIJ1234"	[A-Z]{1,3} first field, repeat [0-9]+ second field	"ABC" - first field [1] "DEF" - first field [2] "GHI" - first field [3] "J" - first field [4] "1234" - second field (the repeating field is terminated when the data "1234" no longer matches the data pattern specified for the first field.)

The example below shows three-field pattern matching.



```

Message definition:
Complex type: Data Element Separation=Use Data Pattern
Field1: xsd:string minOccurs=1, maxOccurs=1, Length=5, Pad=SPACE,
  Data Pattern=".{5}"
Field2: xsd:int minOccurs=0, maxOccurs=1,
  Data Pattern="[0-9]{0,6}"
Field3: xsd:string minOccurs=1, maxOccurs=1, minLength=3, maxLength=4,
  Data Pattern="[A-Z][A-Za-z0-9]{2,3}"

Input1: "ABCDE123F12"
Result1: Field1="ABCDE", Field2="123", Field3="F12"

Input2: "ABCDEF12"
Result2: Field1="ABCDE", Field2=not present, Field3="F12"

Input3: "ABCDE123456XXXX"
Result3: Field1="ABCDE", Field2="123456", Field3="XXXX"

Input4: "ABCDE1234567"
Result4: Field1="ABCDE", Field2="123456", Field3=not present,
  which causes an exception if validation is enabled. One
  character ("7") remains unassigned to any element, which
  also causes an exception.

```

In the case of a repeating child, instances of the child are parsed for as many times as the pattern is matched. This is applied even if *Max Occurs* is specified for the repeating element and the number of occurrences exceeds the upper bound. Therefore some terminating condition must be determinable from the regular expression pattern for the element. The table above includes an example of a repeating element.

When parsing, the data from the input message that matches the *Data Pattern* and is assigned to an element is *not* further scanned for delimiters of a higher level complex type. This behavior is similar to that of *Data Element Separation* method Fixed Length. However, you can code a regular expression that will match data to one of a number of possible delimiters.

When writing, if a length is specified for a child, the value is padded as appropriate to that length. This behavior is similar to that of *Data Element Separation* method Variable Elements Delimited, but without delimiters.

If the message includes a complex type that has *Composition* set to Choice, you can set the *Data Element Separation* method to Use Data Pattern. In this case, the *Data Pattern* values of the children are used to resolve the choice. Starting with the first child, the first pattern to provide a match determines which child is present. Therefore the order of children in a choice might be important.

A complex type can contain repeating children with *Max Occurs* unbounded. Length, and other associated properties such as justification and padding, can optionally be specified for the children.

See “TDS message model integrity” on page 692 for rules that you must follow when using the *Data Element Separation* method Use Data Pattern, and refer to “Combinations of Composition and Content Validation” on page 280 for details of valid settings of *Composition* and *Content Validation*.

### Regular expression syntax:

A regular expression is a coded string. It defines a set of strings that match the expression. A regular expression can be made up of one or more branches (choices), each of which can be a string made up of characters, character classes, or parenthesized expressions with modifiers to specify repetition rules.

The regular expression syntax supported is a subset of XML Schema regular expressions. For the full syntax, see Appendix F in XML Schema Part 2: Datatypes that can be found on the World Wide Web Consortium (W3C) Web site.

The following table lists the supported regular expression syntax elements.

<b>Metacharacter</b>	<b>Meaning</b>
\	escape
.	any single character
*	preceding character 0 or more times
+	preceding character 1 or more times
?	preceding character 0 or 1 time
{...}	occurrences of preceding <sup>1</sup>
[...]	match one of the class contained
[^...]	match one of the class not contained <sup>1</sup>
(...)	group the expressions <sup>1</sup>
	match either preceding or following
<b>Escape sequence</b>	<b>Meaning</b>
\n	new line
\r	carriage return
\t	tab
\e	escape
<b>Class code</b>	<b>Meaning</b>
\d	digit [0-9]
\D	non-digit [^0-9] <sup>2</sup>
\s	whitespace [ \t\n\r]
\S	non-whitespace [^\t\n\r] <sup>2</sup>
\p{L}	all letters <sup>3</sup>
\p{N}	all numbers, similar to \d <sup>4</sup>
[\p{N}\p{L}]	all numbers and all letters , similar to \w <sup>4</sup>
\P{L}	not letters, equivalent to [^\p{L}]
\P{N}	not numbers, equivalent to [^\p{N}]
<b>Range</b>	<b>Meaning</b>
{n}	exactly n times
{n,}	at least n times
{n,m}	at least n but no more than m times
{0,m}	zero to m times

**Notes:**

1. The ellipsis (...) is used to indicate anything inside the { }, or [ ], or ( ) characters.
2. The caret (^) means "not" when inside the [ ] characters.
3. Consult Appendix F of the document XML Schema Part 2: Datatypes for other characters that can be used in place of L and N.
4. Consult Appendix F of the document XML Schema Part 2: Datatypes for the precise differences.

The following table gives some examples of the syntax rules for regular expression syntax. See "Using regular expressions to parse data elements" on page 695 for some examples of their use.

Regular expression data pattern	Meaning
a	Match character "a"
.	Match any one character
a+	Match a string of one or more "a"
a*	Match a string of zero or more "a"
a?	Match zero or one "a"
a{3}	Match a string of exactly three "a", that is "aaa"
a{3,}	Match a string of three or more "a"
a{2,4}	Match a string with a minimum of two and a maximum of four occurrences of "a"
[abc]	Match any one of the characters "a", "b", or "c"
[a-zA-Z]	Match any one character in the range "a" to "z", or in the range "A" to "Z". Note that the range of characters matched is based on the Unicodes of the characters specified.
[^abc]	Match any character except one of "a", "b", or "c"
(ab)+	Match one or more repetitions of the string "ab"
(ab) (cd)	Match either of the strings "ab" or "cd"

**Using multiple delimiters:**

If you set *Data Element Separation* to the method Use Data Pattern, messages in which fields are delimited by one of a set of characters or strings can be parsed.

For example, consider a simple message with two numeric fields that can have either of the characters ';' or '/' delimiting them. There are two approaches that you could use:

1. Model the delimiter as a data element which is added to the message tree. If the message is rewritten, it looks like the input message.

Consider this model:

```

Composition = Sequence
Data Element Separation = Use Data Pattern
  FieldA   Data Pattern = [0-9]*
  Delim    Data Pattern = [;/] optionally with a default value.
  FieldB   Data Pattern = [0-9]*

```

After parsing, the elements FieldA and FieldB each contain any number of the digits 0 to 9, and the element Delim contains either ";" or "/".

2. Recognize the delimiter as a delimiter, which is not added to the tree. If the message is rewritten, a preferred delimiter (as specified in the model) is used.

Consider this model:

```

Composition = Choice
Data Element Separation = Use Data Pattern
  SubType1 Data Pattern = [0-9]*;[0-9]*
  (Composition = Sequence
  Data Element Separation = All Elements Delimited
  Delimiter = ';')
  FieldA
  FieldB
  SubType2 Data Pattern = [0-9]*/[0-9]*
  (Composition = Sequence
  Data Element Separation = All Elements Delimited
  Delimiter = '/')
  FieldA
  FieldB

```

The regular expressions differentiate between the two options that can occur in the message, which are then parsed as a normal delimited structure. After parsing, the elements FieldA and FieldB each contain any number of the digits 0 to 9. The delimiter found in the input message is not saved in an element.

You could refine this approach by using different names for the children, or elements for SubType1 and SubType2, to provide the knowledge of which delimiter is used, or to control which delimiter is included in the output message.

### Using a variable number of repeats:

You can use the *Data Element Separation* method Use Data Pattern to support a variable number of repetitions in an otherwise fixed length environment. However, it relies on the ability to recognize the end of the repetitions based on the data content.

In its simplest form, you can do this by specifying a regular expression Data Pattern that matches a fixed number of characters that is terminated by reaching the end of the message bit stream.

For example, consider a message with one fixed length field (length 10), followed by another fixed length field (length 20) that repeats indefinitely to the end of the bit stream:

```

Message  Data Element Separation=Use Data Pattern
  FieldA Data Pattern=. {10}
  FieldB Repeat, Min Occurs=1, no Max Occurs, Data Pattern=. {20}

```

The following example message contains a fixed length field (length 20) that repeats a variable number of times, and is separated from a second field by the string ";". The pattern specifies a string of 20 characters starting with anything except a semicolon:

```
Message Data Element Separation=All Elements Delimited, Delimiter=;
SubType1 Data Element Separation=Use Data Pattern
FieldA Repeat, Min Occurs=1, no Max Occurs, Data Pattern=[^;].{19}
FieldB
```

### Performance considerations when using regular expressions:

You should take care when specifying regular expressions: some forms of regular expression can involve a large amount of work to find the best match, adversely impacting performance. Other expressions might produce a result that you did not expect.

For example, to match text up to and including a delimiter character ';' do *not* use the pattern ".\*;" because this matches up to the *last* ';' character in the message, including any prior ';' characters in the matched text. Instead, you should use the pattern "[^;]\*;".

Similarly, avoid using the pattern ".\*" because this will always force a search to the end of the message to try and find the best match, and this might result in poor performance. However, you *should* use the pattern ".\*" if you intend to match all remaining data in a message.

For best performance, avoid expressions with redundant nested repeats, such as "([0-9]+)\*". Try to keep the expressions simple, with precise matching criteria. This avoids the need to perform multiple searches for the best match.

## DateTime formats

When you create a `dateTime` object in the MRM, you must specify a format string in the object's *Format String* property for each physical format layer (CWF, TDS, XML). You can use the symbols defined in the information below to control the format in which the `dateTime` appears in the message data.

You can only use `dateTime` for Gregorian calendar dates.

`DateTime` information can appear in a message as:

- String data. This includes TDS, XML, and all CWF physical formats except those mentioned below. This is described further in "DateTime as string data."
- Binary data. This is for the CWF Binary physical format. See "DateTime as CWF BINARY data" on page 707 for more information.
- An offset from an epoch in seconds or milliseconds. This is used if you have set the CWF *Physical Type* property to Time Seconds or Time Milliseconds respectively. See "DateTime as CWF encoded values" on page 708 for details of this option.

The defaults that are set for each message set property that relates to `dateTime`, for each physical representation (CWF, TDS, XML), are defined in "Message set defaults" on page 709.

### DateTime as string data

This section gives information on how you can specify the `dateTime` format using a string of pattern letters.

When you are converting a date or time into a string, a format pattern can be applied that directs the conversion. This would apply if you were formatting from a date or time into a string or parsing a string into a date or time.

During the formatting (for example, a `dateTime` to a string) a pattern or a set of tokens is replaced with their equivalent source. Figure 1 gives a representation of how this is applied.

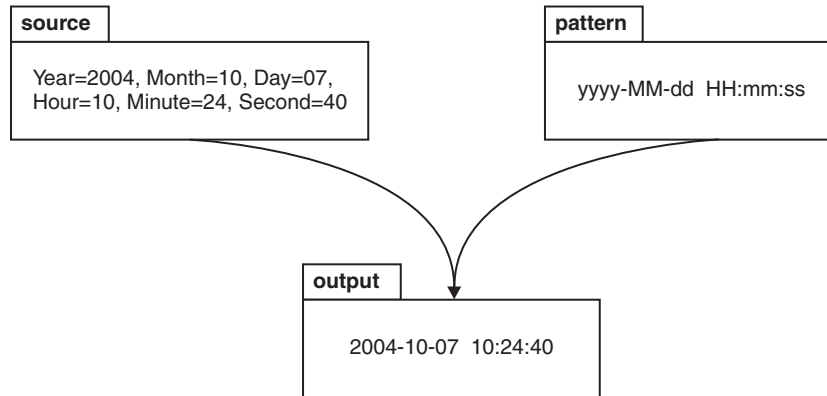


Figure 3. Using a pattern to format a `dateTime` source to produce a string output.

When a string is parsed (for example, converting it to a `dateTime`), the pattern or set of tokens are used to determine which part of the target `dateTime` is represented by which part of the string. Figure 2 gives a representation of how this is applied.

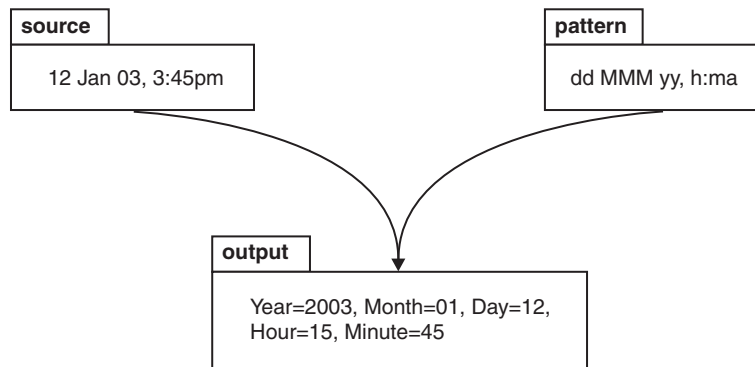
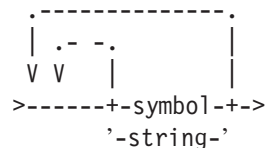


Figure 4. Using a pattern to parse a string source to produce a `dateTime` output.

## Syntax

The expression pattern is defined by:



Where:

**symbol**

is a character in the set **adDeEFGhHIkKmMsSTUwWyYzZ**.

**string** is a sequence of characters enclosed in single quotation marks. If a single quote is required within the string, two single quotes, `''`, can be used.

**Characters for formatting a date`Time` as a string**

The following table lists the allowable characters that can be used in a pattern for formatting or parsing strings in relation to a `dateTime`.

Symbol	Meaning	Presentation	Examples
a	am or pm marker	Text	Input am, AM, pm, PM. Output AM or PM
d	day in month (1-31)	Number	1, 20
dd	day in month (01-31)	Number	01, 31
D	day in year (1-366)	Number	3, 80, 100
DD	day in year (01-366)	Number	03, 80, 366
DDD	day in year (001-366)	Number	003
e	day in week (1-7)	Number	2 <sup>6</sup>
EEE	day in week	Text	Tue
EEEE	day in week	Text	Tuesday
F	day of week in month (1-5)	Number	2 (for second Wednesday in July) <sup>3</sup>
G	Era	Text	BC or AD
h	hour in am or pm (1-12)	Number	6
hh	hour in am or pm (01-12)	Number	06
H	hour of day in 24 hour form (0-23)	Number	7 <sup>7</sup>
HH	hour of day in 24 hour form (00-23)	Number	07 <sup>7</sup>
I	ISO8601 Date/Time (up to yyyy-MM-dd'T'HH:mm:ss.SSSZZZ)	text	2004-10-07T12:06:56.568+01:00 <sup>5</sup>
IU	ISO8601 Date/Time (as above, but ZZZ with output "Z" if the timezone is +00:00)	text	2004-10-07T12:06:56.568+01:00, 2003-12-15T15:42:12.000Z <sup>5</sup>
k	hour of day in 24 hour form (1-24)	Number	8 <sup>7</sup>
k	hour of day in 24 hour form (01-24)	Number	08 <sup>7</sup>
K	hour in am or pm (0-11)	Number	9
KK	hour in am or pm (00-11)	Number	09
m	minute	Number	4
mm	minute	Number	04
M	numeric month	Number	5, 12

Symbol	Meaning	Presentation	Examples
MM	numeric month	Number	05, 12
MMM	named month	Text	Jan, Feb
MMMM	named month	Text	January, February
s	seconds	Number	5
ss	seconds	Number	05
S	decisecond	Number	7 <sup>8</sup>
SS	centisecond	Number	70 <sup>8</sup>
SSS	millisecond	Number	700 <sup>8</sup>
SSSS	1/10,000 th seconds	Number	7000 <sup>8</sup>
SSSSS	1/100,000 th seconds	Number	70000 <sup>8</sup>
SSSSSS	1/1,000,000 th seconds	Number	700000 <sup>8</sup>
T	ISO8601 Time (up to HH:mm:ss.SSSZZZ)	text	12:06:56.568+01:00 <sup>5</sup>
TU	ISO8601 Time (as above, but a timezone of +00:00 is replaced with 'Z')	text	12:06:56.568+01:00, 15:42:12.000Z <sup>5</sup>
w	week in year	Number	7, 53 <sup>2</sup>
ww	week in year	Number	07, 53 <sup>2</sup>
W	week in month	Number	2 <sup>4</sup>
yy	year	Number	96 <sup>1</sup>
yyyy	year	Number	1996 <sup>1</sup>
YY	year: use with week in year only	Number	96 <sup>2</sup>
YYYY	year: use with week in year only	Number	1996 <sup>2</sup>
zzz	time zone (abbreviated name)	Text	gmt
zzzz	time zone (full name)	Text	Greenwich Mean Time
Z	time zone (+/-n)	Text	+3
ZZ	time zone (+/-nn)	Text	+03
ZZZ	time zone (+/-nn:nn)	Text	+03:00
ZZZU	time zone (as ZZZ, "+00:00" is replaced by "Z")	Text	+03:00, Z
ZZZZ	time zone (GMT+/-nn:nn)	Text	GMT+03:00
ZZZZZ	time zone (as ZZZ, but no colon) (+/-nnnn)	Text	+0300
'	escape for text		'User text'
"	(two single quotes) single quote within escaped text		'o'clock'

The presentation of the dateTime object depends on what symbols you specify as follows:



- **Text.** If you specify four or more of the symbols, the full form is presented. If you specify less than four, the short or abbreviated form, if it exists, is presented. For example, EEEE produces Monday, EEE produces Mon.
- **Number.** The number of characters for a numeric date`Time` component must be within the bounds of the corresponding formatting symbols. Repeat the symbol to specify the minimum number of digits required. The maximum number of digits permitted will be the upper bound for a particular symbol. For example, day in month has an upper bound of 31 therefore a format string of d will allow the values of 2 or 21 to be parsed but will disallow the value of 32 or 210. On output, numbers are padded with zeros to the specified length. A year is a special case, see note 1 in the list below. Fractional seconds are also special case, see note 8 below.
- **Lenient date`Time` checking.** Lenient date`Time` checking. The parser will convert out-of-band date`Time` values to the appropriate in-band value. For example, a date of 2005-05-32 will be converted to 2005-06-01. Output of date`Times` will always adhere to the symbol count. For example, a formatting string of yyyy-MM-dd (where '-' is the field separator) allows one or more characters to be parsed against MM and dd. This would allow invalid dates such as 2005-01-123 and 2005-011-12 to be input. The first value of 2005-01-123 would be output as the date 2005-05-03, and the second value of 2005-011-12 would be output as the date 2005-11-12. The number of the timezone formatting symbol Z will only be applicable to the output date`Time` format. Whitespace will be skipped over.
- Any characters in the pattern that are not in the ranges of ['a'..'z'] and ['A'..'Z'] are treated as quoted text. For example, characters like colon (:), comma (,), period (.), the number sign (hash or pound, #), the at sign (@) and space appear in the resulting time text even if they are not enclosed within single quotes.
- You can create formatting strings that produce unpredictable results, so you must use these symbols with care. For example, if you specify dMyyyy, it is impossible to distinguish between day, month, and year. dMyyyy tells the broker that a minimum of one character represents the day, a minimum of one character represents the month, and four characters represent the year. Therefore 3111999 could be interpreted as 3/11/1999 and 31/1/1999.

**Notes:** The following points explain the notes in the table above:

1. Year is handled as a special case:
  - On output, if the count of y is 2, the year is truncated to 2 digits. For example, if yyyy produces 1997, yy produces 97.
  - On input, for 2 digit years the physical message set property of Start of century for 2 digit years is used to determine the century. For example, if Start of century for 2 digit years is set to 53, year 97 is 1997, year 52 is 2052, and year 53 is 1953.
2. The start of a year usually falls in the middle of a week. If the number of days in that week is less than the value specified by the physical message set property Days in First Week of Year, the week is considered to be the last week of the previous year; hence week 1 starts some days into the new year. Otherwise it is considered to be the first week of the new year; hence week 1 starts some days before the new year. For example, Monday of week 1 in 2004 (2004 01 Monday, where Days in First Week of Year = 4 and First Day of Week = Monday ) using format string YYYY ww EEEE is in fact 29th December 2003. If you use Y, the day of week (E) and week in year (w) are adjusted if necessary to indicate that the date falls in the previous year.

If you use the *y* symbol, the adjustment is not done and unpredictable results could occur for dates around year end. For example, if the string 2002 01 Monday is formatted:

- Monday of week 1 in 2002 using format string YYYY ww EEEE is correctly interpreted as 31st December 2001
- Monday of week 1 in 2002 using format string yyyy ww EEEE is incorrectly interpreted as 30th December 2002

Y should only be used in conjunction with w. If you specify Y without w, the year is ignored. For example, if you specify YYYY-MM-dd to format 1996-03-01 the result is 2005-03-01 because the year input is ignored and the current year is assumed.

3. The 11th July 2001 is the second Wednesday in July and can be expressed as 2001 July Wednesday 2 using format string yyyy MMM EEEE F. This is not the same as Wednesday in week 2 of July 2001, which is 4th July 2001.
4. The first and last week in a month might include days from neighboring months. For example, Tuesday 31st July 2001 could be expressed as *Tuesday in week one of August 2001*, which is 2001 08 1 Tuesday using format string yyyy MM W EEEE.
5. See the section *ISO8601, I and T DateTime tokens*.
6. The day in week field is the numeric offset into a week and will vary depending on the value of the physical message set property First Day of Week. For example, the third day in the week would be Wednesday where a physical message set property First Day of Week set to Monday
7. 24 hour fields may result in an ambiguous time if specified with a conflicting am/pm field.
8. Fractional Seconds The length must implicitly match the number of format symbols on input. The output is rounded to the specified length.
9. Long time zones work best when used in the Continent/City format. Similarly, on Unix systems, the TZ environment variable should be specified using the Continent/City format.

## ISO8601, I and T DateTime tokens

If your dateTime values are compliant with the ISO8601:2000 'Representation of dates and times' standard, you should consider whether it is possible to use the formatting symbols I and T. These match a subset of the ISO8601 standard, specifically:

- The restricted profile as proposed by the W3C at <http://www.w3.org/TR/NOTE-datetime>
- Truncated representations of calendar dates as specified in section 5.2.1.3 of ISO8601:2000
  - Basic format (sub-sections c, e and f)
  - Extended format (sub-sections a, b and d)

These symbols should only be used on their own.

- The I formatting symbol matches any dateTime string conforming to the supported subset.
- The T formatting symbol matches any dateTime string conforming to the supported subset that consists of a time portion only.

On output, following form will be applied depending on the logical datatype:

Logical MRM Datatype	Logical ESQ Datatype	Output Form
xsd:dateTime	TIMESTAMP/GMTTIMESTAMP	yyyy-MM-dd'T'HH:mm:ss.SSSZZZ
xsd:date	DATE	yyyy-MM-dd
xsd:gYear	INTERVAL	yyyy
xsd:gYearMonth	INTERVAL	yyyy-MM
xsd:gMonth	INTERVAL	--MM
xsd:gmonthDay	INTERVAL	--MM-dd
xsd:gDay	INTERVAL	---dd
xsd:time	TIME / GMTTIME	'T'HH:mm:ss.SSSZZZ

**Note:**

- On input both I and T accept '+00:00' and 'Z' to indicated a zero time difference from Coordinated Universal Time (UTC), but on output will always generate '+00:00'. If you require that 'Z' is always generated on output, you should use the alternative IU or TU formatting symbols instead.
- ZZZ will always output '+00:00' to indicate a zero time difference from Coordinated Universal Time (UTC). If you require that 'Z' is always generated on output, you should use the alternative ZZZU form instead.

**Using the input UTC format on output**

An element or attribute of logical type xsd:dateTime or xsd:time that contains a dateTime as a string can specify Consolidated Universal Time (UTC) by using either the Z character or time zone +00:00. On input the MRM parser remembers the UTC format of such elements and attributes. On output you can specify whether Z or +00:00 should appear by using the dateTime format property of the element or attribute. Alternatively you can preserve the input UTC format by checking message set property Use input UTC format on output. If this property is checked, then the UTC format will be preserved into the output message and will override that implied by the dateTime format property.

**Examples**

The following table shows a few examples of dateTime formats:

Format pattern	Result
"yyyy.MM.dd'at'HH:mm:ss ZZZ"	1996.07.10 at 15:08:56 -05:00
EEE, MMM d, "yy"	Wed, July 10, '96
"h:mm a"	8:08 PM
"hh 'o'clock' a, ZZZZ"	09 o'clock AM, GMT+09:00
"K:mm a, ZZZ"	9:34 AM, -05:00
"yyyy.MMMMMM.dd hh:mm aaa"	1996.July.10 12:08 PM

**DateTime as CWF BINARY data**

The count of pattern letters determines the number of bytes used to represent a value. The symbol used in the pattern of letters can only be used in groups of 1, 2, or 4, for example, y, yy, or yyyy.

The following table shows the dateTime symbols for CWF binary data:

Symbol	Meaning	Example
y	year	1996
M	month in year	7
d	day in month	10
H	hour in day (0-23)	13
m	minute in hour	30
s	second in minute	55
S	millisecond	978
X	Ignore on input Pad with zeros on output	

The following example shows the C language structure tm with an integer of four bytes:

```
struct tm
{ int tm_sec;      /* seconds after the minute - [0,59]*/
  int tm_min;     /* minutes after the hour   - [0,59]*/
  int tm_hour;    /* hours since midnight     - [0,23]*/
  int tm_mday;   /* day of the month        - [1,31]*/
  int tm_mon;    /* months since January    - [0,11]*/
  int tm_year;   /* years since 1900        */
  int tm_wday;   /* days since Sunday       - [0,6]*/
  int tm_yday;   /* days since January 1    - [0,365]*/
  int tm_isdst;  /* daylight savings time flag */
};
```

You can format this structure by specifying the string "ssssmmmmHHHHddddMMMM+1yyy+1900XXXXXXXXXXXX". The number of pattern letters determines the number of bytes. There are 36 A-Z characters specified in this pattern, which match the 36 byte structure tm. A field followed by a plus sign (+) has the succeeding numeric characters added to it. Therefore MMMM+1 adds one to the month, yyy+1900 adds 1900 to the year. X expects one byte of input, but ignores its value. On output, it outputs the byte as 0.

## DateTime as CWF encoded values

You can represent a dateTime object with the following physical types:

- TimeSeconds. This is a 4 byte integer that represents the number of seconds since the epoch.
- TimeMilliSeconds. This is an 8 byte integer that represents the number of milliseconds since the epoch.

These types provide a way for c time\_t and Java dateTime representations to be parsed.

The epoch (time 0) is specified by the format string. This has the default value of 1970-01-01T00:00 +00:00. You can specify other epoch strings, which must conform to the format pattern "yyyy-MM-dd'T'HH:mm ZZZ".

## DateTime defaults by logical type

The default values assigned to dateTime property are dependant on the logical type of the property. The following table lists the defaults for each of the logical dateTime types:

Logical Type	Default Format
date	yyyy-MM-dd
dateTime	yyyy-MM-dd'T'HH:mm:ss
gDay	- - -dd
gMonth	- -MM
gMonthDay	- -MM-dd
gYear	yyyy
gYearMonth	yyyy-MM
time	HH:mm:ssZZZ

## DateTime component defaults

Default values are assumed if any part of a dateTime object is not present on input. For example, the formatting string yyyy-MM'T'HH:mm does not contain any information about day in month (d), seconds (s), or milliseconds (S). The table below shows the defaults for all dateTime components:

Component	Default value
Year	1970
Month	First month of year
Day	First day of month
Hour	First hour of day
Minute	Minute 0 of hour
Second	Second 0 of minute
Millisecond	Millisecond 0 of second

## Message set defaults

The table below shows the default dateTime formatting symbols for the different physical message representations:

Message set property	CWF default	TDS default	XML default
<i>Default DateTime Format</i>	See Note 1.	See Note 1.	See Note 1.
<i>Default Time Zone ID</i>	Use Broker Locale <sup>2</sup>	Use Broker Locale <sup>2</sup>	Use Broker Locale <sup>2</sup>
<i>Century Window</i>	53	53 (80 for SWIFT)	53
<i>Days in First Week of Year</i>	4	Use Broker Locale <sup>2</sup>	Use Broker Locale <sup>2</sup>
<i>First Day of Week</i>	Monday	Use Broker Locale <sup>2</sup>	Use Broker Locale <sup>2</sup>

### Note:

1. You can either set the default dateTime format to be derived from its logical type (the default), or specify the dateTime format to be used. This is set at the message set level for each physical format that has been added.
2. The key phrase Use Broker Locale causes the broker to get the information from the underlying platform.

You can update all these default values. The CWF defaults are set for all values of the *Physical Type* property. If you change the CWF *Physical Type* to Binary, Packed Decimal, TimeSeconds, or TimeMilliseconds, you must update the *Default DateTime Format* property manually to ensure consistent results.

For more information about these message set properties, see “Custom Wire Format message set properties” on page 150, “TDS Format message set properties” on page 162, or “XML Wire Format message set properties” on page 155.

## Message model task list errors that have a quick fix

This provides a list of message modeling task list warnings or errors where a quick fix can be applied to correct them.

### Unresolved references

The following table provides a list of those errors that have references that cannot be resolved:

Error type	Description	Quick Fix
Attribute reference error	The attribute reference cannot be resolved	Allows you to add the missing include or import statement
Attribute group reference error	The attribute group reference cannot be resolved	Allows you to add the missing include or import statement
Attribute type reference	The attribute type reference cannot be resolved	Allows you to add the missing include or import statement
Base type error	The type has an unresolved base type	Allows you to add the missing include or import statement
Element reference error	The element reference cannot be resolved	Allows you to add the missing include or import statement
Element type reference error	The element type reference cannot be resolved	Allows you to add the missing include or import statement
Group reference error	The group reference cannot be resolved	Allows you to add the missing include or import statement
Schema directive error	The schema directive cannot be resolved	Allows you to add the missing include or import statement
Sub group error	The element declaration references a head element which cannot be resolved.	Allows you to add the missing include or import statement

### Other errors

The following table provides a list of additional warnings or errors that can be cleared using a quick fix:

Error type	Description	Quick Fix
Message key deprecated warning	TDS property "Message Key" has been superseded by logical property "Message Alias".	Will update your message definition to use "Message Alias" instead. (Recommended only if you have only Version 6.0 brokers in your domain.)
Message key enumeration deprecated warning	TDS property "Interpret Element Value = Message Key" has been superseded by logical property "Interpret Value As = Message Identity".	Will update your element definition to use logical property "Interpret Value As = Message Identity" instead. (Recommended only if you have only Version 6.0 brokers in your domain.)
Repeat count deprecated warning #1	CWF property "Repeat Count" has been superseded by "Max Occurs". Both "Repeat Count" and "Max Occurs" have been set, but do not have the same value.	You will have a choice of two quick fixes: <ul style="list-style-type: none"> <li>• Will update your definition to unset the "Repeat Count" property.</li> <li>• Will update your definition to set "Max Occurs" to the value of the "Repeat Count" property, and to unset the "Repeat Count" property.</li> </ul>
Repeat count deprecated warning #2	CWF property "Repeat Count" has been superseded by "Max Occurs". Both "Repeat Count" and "Max Occurs" have been set and have the same value.	Will update your definition to unset the "Repeat Count" property.
Redefine error	An XML Schema Redefine construct has been found but is not supported.	Will update your message definition file to use an XML Schema Include construct instead. Any redefinitions will be lost.
Value does not match Length facet error	The length of a default value, fixed value or enumeration value does not match the effective Length facet for the simple type.	You will have a choice of two quick fixes: <ul style="list-style-type: none"> <li>• Will update your simple type definition so that the Length facet is converted to a Max length facet.</li> <li>• Will update all the simple type definitions in your message definition file so that all Length facets are converted to Max Length facets, and then save the file to remove all the associated task list errors.</li> </ul>
Facet not applicable for simple type error	A facet has been found on a simple type, but the facet is either not permitted on that simple type or is a duplicate.	Will update your simple type definition so that all illegal facets and all duplicate facets are removed.

---

## Additional MIME domain information

This section provides additional information in relation to the MIME domain. This information is categorized into:

- “MIME standard header fields”
- “MIME Parser Use and Restrictions” on page 715

### MIME standard header fields

This section is a summary of the common MIME headers and may be useful as a quick reference. It is not a definitive specification of MIME. In some cases the MIME parser allows documents which are not strictly valid according to the standard. For example, it does not insist on the presence of a MIME-Version header. All the standard MIME header fields are simply written to the logical tree as they appear in the MIME document. The MIME parser only takes special note of the Content-Type header field.

All MIME headers may include comments enclosed by parentheses as shown in the example for the MIME-Version header.

### MIME header fields

#### MIME-Version

Example:

```
MIME-version: 1.0 (generated by my-application 1.2)
```

For a MIME document to conform with RFC 2045, this field is required in the top-level header with a value of 1.0. MIME-Version should not be specified on individual parts.

#### Content-Type

Content-Type is not required for a document to conform with RFC 2045, but a top-level Content-Type is required by the MIME parser. Content-Type defaults to text/plains. Content-Type defines the type of data in each part as a type/subtype. The MIME parser accepts most values for Content-Type and simply stores them in the logical tree. The only exceptions are:

- The MIME parser rejects any Content-Type value with type = message
- The MIME parser assumes that a Content-Type value with type = multipart introduces a multipart MIME document and rejects such a value if it does not contain a valid boundary parameter. The value of the boundary parameter defines the separator between message parts in a multipart message. In a nested multipart message, a unique boundary value is needed for each nesting level.

Syntax:

```
Content-Type: type/subtype;parameter
```

Where type and subtype define the Content-Type and any optional parameters are delimited by semicolons.

Example 1:

```
Content-Type: multipart/related;type=text/xml
```



In example 1 the Content-Type is defined as multipart/related and also has an optional parameter definition (type=text/xml). While this is syntactically correct, as there is no valid boundary parameter this message will be rejected.

Example 2:

Content-Type: multipart/related;boundary=Boundary;type=text/xml

Example 2 shows a valid Content-Type definition, both in terms of syntax and semantics. The boundary value may optionally be enclosed in quotes. When it appears in the MIME body the value is preceded by the sequence '--' and care must be taken that the resulting value (in this example it would be --Boundary) cannot appear in the message body. If the message data is encoded as quoted-printable, you are recommended to have a boundary that includes a sequence such as "=", which cannot appear in a quoted-printable body.

Some common Content-Type values are shown below. Any other values are allowed and simply stored in the logical tree.

Table 1. MIME Content-Types

Content-Type	Description
text/plain	Generally used for a typical mail or news message. text/richtext also common.
text/xml	Generally used with SwA (SOAP with Attachments)
application/octet-stream	Used where the message is an unknown type and contains any kind of data as bytes.
application/xml	Used for application-specific xml data
x-type	Used for non-standard content type. It must start x-
image/jpeg	Used for images. image/jpeg and image/gif are common image formats that are used
multipart/related	Used for multiple related parts in a message. Specifically used with SwA (SOAP with Attachments)
multipart/signed	Used for multiple related parts in a message including signature. Specifically used with S/MIME
multipart/mixed	Used for multiple independent parts in a message

### Content-Transfer-Encoding

Optional. Many Content-Types are represented as 8-bit character or binary data. This could include XML, which typically uses UTF-8 or UTF-16 encoding. This type of data cannot be transmitted over some transport protocols and may be encoded to 7-bit.

The Content-Transfer-Encoding header field is used to indicate the type of transformation that has been used for encoding this type of data into a 7-bit format.

The only values allowed by the WS-I Basic Profile are:

- 7bit - the default
- 8bit
- binary
- base64
- quoted-printable

The values 7bit, 8bit, and binary all effectively mean that no encoding took place. It is possible that a MIME conformant mail gateway might use this value to control how it handles the message. For example, encoding it as 7bit before passing routing it over SMTP.

The values base64 and quoted-printable mean that the content has been encoded. The value quoted-printable means that only non-7-bit characters in the original are encoded and is intended to yield a document which is still human-readable. This setting is most likely to be used in conjunction with a Content-Type of text/plain.

**Content-ID**

Optional. This enables parts to be labeled and referenced from other parts of the message. These parts are typically referenced from part 0 (the first) of the message.

**Content-Description**

Optional. This enables parts to be described.

**MIME encodings**

The following section is aimed to provide a basic guide to the base64 and quoted-printable encoding. Please refer to RFC 1521 for a definitive specification of MIME encodings.

**base64**

The original data is broken into groups of 3 octets. Each group is then treated as 4 concatenated 6-bit groups, each of which is translated into a single digit in the base64 alphabet. The base64 alphabet is A-Z, a-z, 0-9, and / (with A=0 and /=63).

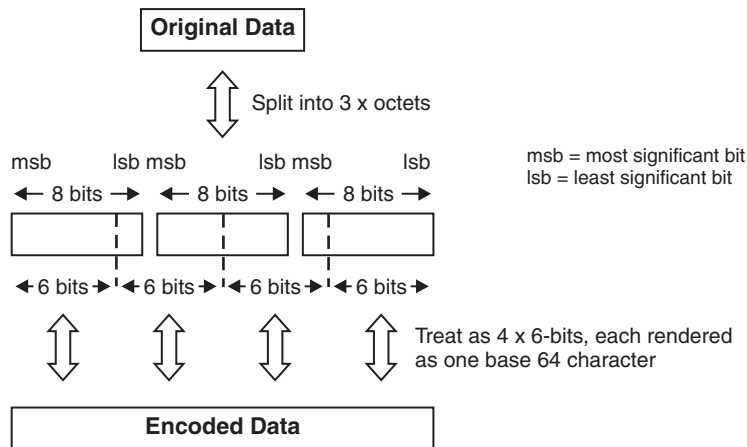


Figure 5. base64 data transformation

If fewer than 24 bits are available at the end of the data, the encoded data is padded using the "=" character. The maximum line length in the encoded data is 76 characters and line breaks (and any other characters not in the alphabet above) are ignored when decoding.

Examples:

Input	Output
Some data encoded in base64.	U29tZSBkYXRhIGVuY29kZWQgaW4gYmFzZTY0Lg==
life of brian	bGlmZSBvZiBicmlhbg==\012
what	d2hhdA==

### quoted-printable

This encoding is only appropriate if most of the data comprises printable characters. Specifically, characters in the ranges 33-60 and 62-126 are usually represented by the corresponding ASCII characters. Control characters and 8-bit data must be represented by the sequence = followed by a pair of hex digits.

The standard ASCII space <SP> and horizontal tab <HT> represent themselves, unless they appear at the end of an encoded line (without a soft line break) in which case the equivalent hex format must be used (=09 and =20 respectively).

Line breaks in the data are represented by the RFC 822 line break sequence <CR><LF> and should be encoded as "=0D=0A" if binary data is being encoded.

As for base64, the maximum line length in the encoded data is 76 characters. An '=' sign at the end of an encoded line (a 'soft' line break) is used to tell the decoder that the line is to be continued.

## MIME Parser Use and Restrictions

This provides a general introduction to the MIME parser and some of the restrictions in its use. The MIME domain does not support the full MIME standard, but is intended to support specific known uses of MIME.

MIME stands for Multipurpose Internet Mail Extensions. A multipart MIME message comprises a number of message parts, each qualified by MIME headers. The MIME domain and parser enable the user to parse and write multipart MIME messages.

MIME has historically been used to send e-mail messages. When e-mail attachments are involved, a multipart MIME message is used. Multipart MIME is becoming more widely used as a convenient physical format for sending other kinds of message that have attachments or consist of multiple separate parts. Examples are:

- RosettaNet. Here each part is typically a separate XML document but there may also be non-XML attachments. Note that the MIME parser enables the parsing of MIME messages of the style used by RosettaNet, including nested multipart messages. However, it does not offer specific support for the wider RosettaNet architecture or PIPs (Partner Interface Processes).
- SOAP with Attachments (SwA). Here the first part is a normal SOAP XML message and the other parts contain XML or non-XML attachments.
- TLOG. This is a specialized use of SwA, where the attachments are groups of point of sale Transaction Log records in either one of two XML forms or a tagged/delimited string form. (Different POS devices generate different TLOG record formats such as ACE. In addition, a store may then process the record before uploading or it may be sent raw.)

## Restrictions

The MIME parser is bitstream driven and has no external metadata. It relies exclusively on bitstream metadata when parsing and on tree metadata when writing. The parser does not validate MIME messages against a message model and it ignores the tooling Validate property. The parts of a MIME message are handled as BLOBs. The user may choose to subsequently (re-)parse specific MIME parts using a different parser, and if this is an MRM parser, then they can be validated as usual. The MIME parser does not support on demand parsing and ignores the Parse Timing property.

The user can either specify the new MIME domain at runtime in an MQRFH2 header (WebSphere MQ only) or statically in their message flow in the tooling (on the input nodes: MQGet, HTTPRequest and ResetContentDescriptor). The MIME parser will then be invoked to own the last child of root (for example, the message body). The MIME domain can be specified with the ESQL CREATE PARSE clause and ASBITSTREAM function to parse and write bitstreams. The MIME parser handles documents received both over the HTTP transport (where the Content-Type appears as an HTTP header) and over other transports (where the Content-Type header is part of the message body). In both cases the value of the Content-Type header will be managed using the broker ContentType property.

In practice it is expected that the MIME parser will handle the majority of uses of MIME in application-to-application messaging, including multipart MIME with a single part and non-multipart MIME documents.

---

## Generated model representations

This section provides information on the possible generated model representations. Details are provided for:

- "Document generation"
- "WSDL generation"
- "XML Schema generation" on page 719

### Document generation

#### Output Files

The document generator produces a set of HTML pages and any necessary files (for example, images) that are required to display the pages correctly.

There is one page for each message definition file in the message set, and one additional index page linking these pages together.

The index page (index.html), is intended to be the "entry point" into the documentation.

### WSDL generation

This topic defines the objects created by the WSDL Generator.

#### Generated Files

The default file and definition element names are shown in the table below. <Message Set> is the supplied message set name and <Definition Name> is the

supplied Definition Name solicited by the wizard.

Table 2. WSDL File Naming Convention

File	File Name	File Extension	Value of name attribute on WSDL <definitions> element
Service File (multi- or single-file format)	<Message Set>Service	wsdl	<Definition Name>Service
Binding File	<Message Set>Binding	wsdl	<Definition Name>Binding
Interface File	<Message Set>Interface	wsdl	<Definition Name>
Schema File(s)	Message Definition File	xsd	not applicable

## Report File

The WSDL generator appends the result of the generation operation to a report file, listing any errors which occurred. The file name is:

<Message Set>.wsdlgen.report.txt

## WSDL Content

The tables below show the element / attribute values to be set in the generated WSDL. The elements are described top-down as they appear in a conventionally ordered WSDL document. The <schema> section of the WSDL definition is not shown since this corresponds directly to the broker message definitions.

Element names are from the WSDL 1.1 namespace except where prefixed by soap: for the WSDL SOAP namespace. Operation elements occur in both the binding and portType sections, so operation is qualified as necessary – e.g. portType / operation.

The following values apply to the WSDL definition as a whole:

Table 3. WSDL objects

Element	Attribute	Value
definitions	xmlns	assign namespace prefixes.
definitions	targetNamespace	<Company Domain Name>/<Message Set>/definitions where <Company Domain Name> is solicited by the wizard and <Message Set> is the message set name.
message	name	<operation>-<role> where <operation> is the associated category name and <role> is input, output or fault
part	name	name of the broker message M. If Style is set to rpc, the body parts are defined using the type attribute. If not, the body parts are defined using the element attribute.
portType	name	<Message Set>

Table 3. WSDL objects (continued)

Element	Attribute	Value
binding	name	<ul style="list-style-type: none"> <li>• "&lt;Message Set&gt;SOAP_HTTP_Binding"</li> <li>• "&lt;Message Set&gt;SOAP_JMS_Binding"</li> </ul>
soap:binding	style	From the value of Style set in the New WSDL definition wizard.

The following values apply to each individual WSDL operation (a WSDL operation is represented as a message category):

Table 4. WSDL <operation> objects

Element	Attribute	Value
operation	name	The name of the message category
soap:operation	style	From the value of Style set in the New WSDL definition wizard.
input, output	name	This optional attribute is not generated
fault	name	operation-fault-part
soap:body	namespace	<ul style="list-style-type: none"> <li>• If Style has been set to rpc then it is the namespace of the corresponding broker message.</li> <li>• If Style has been set to document the attribute is not generated.</li> </ul>
soap:body	use	If the xsi:type Output Policy on the message is set to for all elements, this will be set to <i>encoded</i> . If the Output Policy is set to anything else, this will be set to <i>literal</i> .
soap:header, soap:fault	use	This is set to <i>literal</i> .
portType / operation	parameterOrder	<ul style="list-style-type: none"> <li>• If the Style is set to document, the attribute is not generated.</li> <li>• If the Style is set to rpc, the input parts are I1..IN and output parts O1..ON. This is a space-separated list of parts omitting any explicitly identified return parts (or O1 if no return part is specified). For example: I1 I2 I3 O2 O3 O4</li> </ul>

## Message Set

The main result of generating WSDL from a message set is a WSDL definition for consumption by an external toolkit such as .NET. Typically, however, the broker will also need to parse the runtime SOAP messages described by that WSDL. By default the WSDL generator will leave your message set SOAP-ready for use by the MRM XML parser. This section describes what that means.

- The required SOAP mxsds are added to the message set.
- If the specified WSDL style is set to rpc, the WSDL generator also creates an mxsd in the message set with a target namespace defined by the RPC namespace that was set in the New WSDL definition wizard.

- The name of the mxsd is <Message Set>-rpc.mxsd where <Message Set> is the name of the message set.
- The mxsd contains new messages for each WSDL operation (in other words for each of the message categories).
- If the operation (that is the category name) is operation1 then:
  - If there are one or more messages  $P_1..P_m$  with Role Type wsdl:input and Role Usage soap:body then generate a message operation1 of a local complex type. This is a sequence of no-namespace elements  $P_1..P_m$ . Each of the elements  $P_1..P_m$  is of a complex type, as defined for the corresponding message  $P_x$  and referred to by the Category.
  - If there are one or more messages  $P_1..P_m$  with Role Type wsdl:output and Role Usage soap:body then generate a message operation1Response of a local complex type. This is a sequence of no-namespace elements  $P_1..P_m$ . Each of the elements  $P_1..P_m$  is of a complex type, as defined for the corresponding message  $P_x$  and referred to by the Category.

## Validation Annotation

The generated WSDL is annotated to state whether it is compliant with the Basic Profile. The WSDL generator emits the following claim on the wsdl:port element or on the highest level element for which conformance is being claimed:

```
<wsdl:definitions etc.
  xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/"\>

  <wsdl:port name="etc" binding="etc">
    <wsdl:documentation>
      <wsi:Claim conformsTo="http://ws-i.org/profiles/basic/1.0"/>
    </wsdl:documentation>
```

## XML Schema generation

This topic covers the behavior of XML Schema generation. For example, you could use the schema generated from a message definition file to subsequently validate XML instance documents written by WebSphere Message Broker.

### Lax generation

Lax generation affects how complex types that have *Content Validation* set to Open or OpenDefined or have *Composition* set to UnorderedSet are rendered in the generated schema. Note that such a validating schema will permit a wider range of messages than MRM parser validation.

#### Content Validation is set to Open or OpenDefined

Here a complex type (global or anonymous) has its content replaced by a single element of type anyType. The following generation pattern is used for complex types with *Content Validation* set to Open:

```
<element name="xmlNameOfMessage">
  <complexType>
    <sequence>
      <any processContent="lax"
        minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>
```

Where *Content Validation* is set to *OpenDefined*, the following pattern is used. (The namespaces listed are all those defined in the containing message set.)

```
<element name="xmlNameOfMessage">
  <complexType>
    <sequence>
      <any processContent="lax"
          minOccurs="0" maxOccurs="unbounded"
          namespace="http://www.ns1 http://www.ns2" />
    </sequence>
  </complexType>
</element>
```

### Composition is set to *UnorderedSet*

Where *Composition* is set to *UnorderedSet*, to mimic the unordered aspect, a choice is inserted with appropriate cardinality. This is shown below.

```
<element name="xmlNameOfMessage">
  <complexType>
    <sequence maxOccurs="unbounded"
        minOccurs= "(minOccurs of original sequence) *
            (items in original sequence)">
      <choice>
        .. sequence contents ..
      </choice>
    </sequence>
  </complexType>
</element>
```

### Strict generation

Strict generation affects how complex types that have *Content Validation* set to *Open* or *OpenDefined* or have *Composition* set to *UnorderedSet* are rendered in the generated schema. Note that such a validating schema will permit a narrower range of messages than MRM parser validation.

Strict is the default generation option and generates a schema that matches the schema held in the message definition file, without the model extensions.

#### Content Validation set to *Open/OpenDefined*

A complex type (global or anonymous) will lose the ability to contain self-defining elements and becomes closed.

#### Composition set to *UnorderedSet*

A complex type (global or anonymous) will lose the ability to be unordered and becomes a sequence.

### Rendering of `xsd:elements`

If an XML physical format is specified when generating the schema, the wire format customization is applied to the logical model. These properties control how an element in the model is actually rendered when it appears in a message for an XML wire format. See “XML rendering options” on page 678 for the different render options available. A generated schema example is given below showing what is generated for the different render options available for local elements; note these examples do not modify the Namespace of any ID Attribute Name or Value Attribute Name properties and assume that all elements specified in the `complexType1` are of schema built-in type string.

```
<xsd:complexType name="complexType1">
  <xsd:sequence>
    <!-- Local element Render = 'XMLElement' -->
```



```

    <xsd:element name="localElement1" type="xsd:string"/>
<!-- Local element Render = 'XMLElementAttrID'
      ID Attribute Name = 'id' -->
    <xsd:element name="localElement2">
      <xsd:complexType>
        <xsd:simpleContent>
          <xsd:extension base="xsd:string">
            <xsd:attribute name="id" type="xsd:string"/>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
<!-- Local element Render = 'XMLElementAttrVal'
      Val Attribute Name = 'val' -->
    <xsd:element name="localElement3">
      <xsd:complexType>
        <xsd:simpleContent>
          <xsd:extension base="xsd:string">
            <xsd:attribute name="val" type="xsd:string"/>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
<!-- Local element Render = 'XMLElementAttrIDVal'
      ID Attribute Name = 'id' Val Attribute Name = 'val' -->
    <xsd:element name="localElement4">
      <xsd:complexType>
        <xsd:simpleContent>
          <xsd:extension base="xsd:string">
            <xsd:attribute name="val" type="xsd:string"/>
            <xsd:attribute name="id" type="xsd:string"/>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  <!-- Local element Render = 'XMLAttribute' -->
  <xsd:attribute name="localElement5" type="xsd:string"/>
</xsd:complexType>

```

## Rendering of xsd:attributes

The rendering of xsd:Attributes is not supported. The user can only change the name of the attribute.

## Embedded simple types and Compound Elements

These are modeled in the message definition file as elements with both *minOccurs* and *maxOccurs* set to 0 and have one of the predefined ComIbmMrm\_XXX types. During the schema generation, the type of such elements is changed to the base type of the respective ComIbmMrm\_XXX type.

If there are global simple types that inherit from one of these ComIbmMrm\_XXX types, these are changed to inherit from the base type of the corresponding ComIbmMrm\_XXX type.

Generated schema files will not have any occurrence of these ComIbmMrm\_XXX types.

For example the global element with type defined below:

```
<element name="globalElement1" type="ns1:ComIbmMrm_BaseValueBinary"/>
```

will result in the generated schema file and a global element with the corresponding xsd base type as defined below:

```
<element name="globalElement1" type="hexBinary"/>
```

## Import formats

This section provides information on the supported features of formats that have been imported from an external source. Details are provided for:

- “Importing from C: supported features”
- “Importing from COBOL: supported features” on page 724
- “Importing from WSDL: generated objects and restrictions” on page 729
- “Importing from XML Schema: unsupported features” on page 732
- “Importing from XML DTD: unsupported features” on page 734

### Importing from C: supported features

This section describes the defaults that the C importer uses when mapping C data types to MRM data types.

The table below shows how the C definitions influence the XML schema settings in the message model. Some xsd types have '-' after the type. This indicates that it is an anonymous simple type based on this type. For strings, the purpose of the anonymous type is to add a length restriction; for numeric types the purpose is to add a minimum and/or maximum value restriction.

The Packed data structure data type is outside the scope of the C importer.

C data type	Logical type	Notes
Char	xsd:string-	length=1
Char[10]	xsd:string-	length=10
Char[10][3]	xsd:string-	length=3
Char[10][3][6]	xsd:string-	length=6
Unsigned Char	xsd:unsignedByte	
Unsigned Char[2]	xsd:unsignedByte	
Signed Char	xsd:byte	
Signed Char[2]	xsd:byte	
Int	xsd:int	
Int[2]	xsd:int	
Int[2][3]	xsd:int	
Unsigned Int	xsd:unsignedInt	
Short	xsd:short	
Unsigned Short	xsd:unsignedShort	
Long	xsd:int	
Long Long Int	xsd:long	
Float	xsd:float	
Double	xsd:double	
Long Double <small>(see note 1)</small>	xsd:double	
<any pointer type>	xsd:hexBinary-	length= <small>(see note 2)</small>

C data type	Logical type	Notes
<any enum>		(see note 3)

The following table shows how C definitions influence the physical CWF characteristics of the elements generated in the message model.

C data type	CWF Physical type	CWF Length characteristics	Other CWF characteristics
Char	Fixed Length	Length Count = 1 Length Units = Bytes	
Char[10]	Fixed Length	Length Count = 10 Length Units = Bytes	Left justify
Char[10][3]	Fixed Length	Length Count = 3 (and Max Occurs = 10) Length Units = bytes	Left justify
Char[10][3][6]	Fixed Length	Length Count = 6 (and Max Occurs = 30) Length Units = bytes	Left justify
Unsigned Char	Integer	Length Count = 1	Signed = no
Unsigned Char[2]	Integer	Length Count = 1 (and Max Occurs = 2)	Signed = no
Signed Char	Integer	Length Count = 1	Signed = yes
Signed Char[2]	Integer	Length Count = 1 (and Max Occurs = 2)	Signed = yes
Int	Integer	Length Count = 4	Signed = yes
Int[2]	Integer	Length Count = 4 (and Max Occurs = 2)	Signed = yes
Int[2][3]	Integer	Length Count = 4 (and Max Occurs = 6)	Signed = yes
Unsigned Int	Integer	Length Count = 4	Signed = no
Short	Integer	Length Count = 2	Signed = yes
Unsigned Short	Integer	Length Count = 2	Signed = no
Long	Integer	Length Count = 4 (see note 4)	Signed = yes
Long Long Int	Integer	Length Count = 8	Signed = yes
Float	Float	Length Count = 4	
Double	Float	Length Count = 8	
Long Double (see note 1)	Float	Length Count = 8	
<any pointer type>		(see note 2)	
<any enum>	Integer	(see note 3)	

#### Notes:

1. Do not set the value of C importer option size of long double to 128 bit. This will not import successfully. Please use the default 64 bit.
2. The length count is affected by the Address Size C importer option as follows:
  - For 32 bit, CWF length count = 4 bytes.
  - For 64 bit, CWF length count = 8 bytes.
3. The type and length of an enum is affected by the Size of enum C importer option as follows:
  - For 1: Logical type = xsd:byte, CWF physical type = Integer, CWF length count = 1 byte.
  - For 2: Logical type = xsd:short, CWF physical type = Integer, CWF length count = 2 bytes.
  - For 4: Logical type = xsd:int, CWF physical type = Integer, CWF length count = 4 bytes.
  - For Compact: The smallest representation is chosen that the enumeration fits into.
4. The length of a long is affected by the Address Size C importer option as follows:
  - For 32 bit: CWF length count = 4 bytes.
  - For 64 bit: CWF length count = 8 bytes.
5. Element names that clash with Java language keywords are modified by prefixing them with a single underscore character.
6. The `_Packed` keyword is not supported. Only ANSI C declarations are supported.
7. The C long long data type is not supported.
8. C++ object oriented extensions are not supported. Only ANSI C declarations are supported.
9. Pointers will be imported as xsd:integer with CWF length count set to 4.
10. Recursive C structures are not supported. If a nested structure contains a structure with a name that is the same as the parent structure, the import succeeds but the logical definitions are not correct. To avoid this problem, ensure that the name of the nested structure is not the same as that of the outer or parent structure.

## Importing from COBOL: supported features

The following table shows how COBOL definitions influence the XML schema settings in the message model. Note that some xsd types have '-' after the type. This indicates that it is an anonymous simple type based on this type. For strings, the purpose of the anonymous type is to add a length restriction; for numeric types the purpose is to add a minimum and/or maximum value restriction.

COBOL Clause	XML Schema datatype	Notes
PIC A	xsd:string -	
PIC X	xsd:string -	
PIC 9(n) n = 1-4	xsd:short -	DISPLAY, COMP, or COMP-3
PIC 9(n) n = 5-9	xsd:int -	DISPLAY, COMP, or COMP-3
PIC 9(n) n = 10-18	xsd:long -	DISPLAY, COMP, or COMP-3

COBOL Clause	XML Schema datatype	Notes
PIC 9(n) n = 19-31	xsd:integer -	DISPLAY, COMP, or COMP-3
PIC 9(n)V9(m)	xsd:decimal -	DISPLAY, COMP, or COMP-3 any virtual decimal point value
COMP-1	xsd:float -	
COMP-2	xsd:double -	
Any edited string	xsd:string -	
Any edited number	xsd:string -	For example, PICZ
VALUE	All	Non-88 Level VALUE clauses can be imported as schema default values (option on import wizard).

The following table shows how COBOL definitions influence the physical CWF characteristics of the elements generated in the message model.

COBOL Keyword	CWF Physical Type	CWF Length Characteristics	Other CWF characteristics
PIC X(n) PIC A(n)	Fixed Length String	Length = n Length Units = Bytes	Justification = Left Justify Padding Character = SPACE
PIC 9(n) DISPLAY n=1-9	External Decimal	Length = n Length Units = Bytes	Justification = Right Justify Padding Character = '0' Signed = Unticked Sign Orientation = Trailing
PIC 9(n) DISPLAY n=10-31	External Decimal	Length = n Length Units = Bytes	Justification = Right Justify Padding Character = '0' Signed = Unticked Sign Orientation = Trailing
PIC 9(n) COMP, COMP-4, COMP-5 or BINARY	Integer	Length = 1, 2, 4 or 8 based on n Length Units = Bytes	Signed = Unticked Sign Orientation = Blank
PIC 9(n) COMP-3 n=1-9	Packed Decimal	Length = CEILING((n+1)/2) Length Units = Bytes	Signed = Unticked Sign Orientation = Blank
PIC 9(n) COMP-3 n=10-31	Packed Decimal	Length = CEILING((n+1)/2) Length Units = Bytes	Signed = Unticked Sign Orientation = Blank
PIC S9(n) DISPLAY n=1-31	External Decimal	Length = n Length Units = Bytes	Signed = Unticked Sign Orientation = Blank *See Note 1
PIC S9(n) COMP or COMP-3 n=1-31	Integer or Packed Decimal	Length = See COMP and COMP-3 definitions above Length Units = Bytes	Signed = Unticked Sign Orientation = Blank

COBOL Keyword	CWF Physical Type	CWF Length Characteristics	Other CWF characteristics
PIC 9(m)V9(n) DISPLAY n=1-31	External Decimal	Length = n+m  Length Units = Bytes	Signed = Unticked  Sign Orientation = Trailing  Virtual Decimal Point = n
PIC 9(m)V9(n) COMP or COMP-3	Integer or Packed Decimal	Length = CEILING((n+m+1)/2) for COMP-3  Length = 1, 2, 4 or 8 for COMP  Length Units = Bytes	Signed = Unticked  Sign Orientation = Trailing
COMP-1	Float	Length = 4  Length Units = Bytes	Signed = Ticked  Sign Orientation = Blank
COMP-2	Float	Length = 8  Length Units = Bytes	Signed = Ticked  Sign Orientation = Blank
SYNC	Float, Integer or Packed Decimal		Leading Skip Count as appropriate  Trailing Skip Count as appropriate  Byte alignment as appropriate  *See note 2

COBOL Keyword	CWF Physical Type	CWF Length Characteristics	Other CWF characteristics
<p><b>Notes:</b></p> <ol style="list-style-type: none"> <li>1. Sign Orientation can take one of the following values, based on the SEPARATE, LEADING or TRAILING keywords in the COBOL definition. <ul style="list-style-type: none"> <li>• Leading</li> <li>• Leading Separate</li> <li>• Trailing</li> <li>• Trailing Separate</li> </ul> </li> <li>2. SYNC Keyword causes the field to be aligned on a 1, 2, 4, or 8 byte boundary. This may cause 'slack bytes' to be added either before or after a field. Leading Skip Count is the number of such bytes added before; Trailing Skip Count is the number added after. <p>Leading Skip Count and Trailing Skip Count are calculated for each of the imported elements by the importer irrespective of SYNC clause. They will have non zero values when the SYNC clause is present.</p> <p>Where there is a repeating element, Leading Skip Count and Trailing Skip Count are used for the first occurrence of the repeating element, for subsequent occurrences, only the Trailing Skip Count is used.</p> <p>Refer to COBOL reference material for details of fields requiring Byte Alignment.</p> </li> <li>3. The COBOL importer requires all files you are importing to be syntactically correct. Results are unpredictable if this is not the case.</li> <li>4. COBOL data types including POINTER, COMP-X, INDEX and PROCEDURE-POINTER are not supported.</li> <li>5. COBOL containing the keyword NATIVE causes an error and will not import.</li> <li>6. COBOL level 66 and level 77 data items are not imported.</li> <li>7. Hexadecimal binary values cannot be attributed to non-numeric literals. They cannot reside in the LINKAGE SECTIONS that are imported by the COBOL importer. They can reside elsewhere in the COBOL file. Alternatively, you can convert the hexadecimal value to a char string for PIC X, or to a decimal number for PIC 9.</li> <li>8. Where element names clash with Java language keywords, they are modified by prefixing the element name with a single underscore character.</li> <li>9. Object-oriented extensions to COBOL 85 are not supported. For example, OBJECT-REFERENCE is not supported.</li> <li>10. COBOL OCCURS DEPENDING ON clause. The Byte Alignment, Leading Skip Count, and Trailing Skip Count CWF properties of elements within such a structure are not set up properly. You must correct these using the message editor.</li> <li>11. When the imported COBOL source file contains QUOTE or QUOTES in the value clause of a picture string, the default behavior is to fill in the data with double quotation marks unless you set the COBOL QUOTE compile option to SINGLE on the Import Options page of the COBOL importer wizard.</li> </ol>			

## Signed external decimal numbers

The Custom Wire Format (CWF) component of the WebSphere Message Broker provides support for the modelling of numeric data using the External Decimal (also known as Zoned Decimal) data format. In this format, a number is stored internally as decimal character data. For example, in a system using the EBCDIC code, the number 1234 stored in a 4-byte external decimal field would be stored as the character string "1234" and its actual internal hexadecimal representation would be F1F2F3F4.

With signed external decimal numbers, the sign can be incorporated into the actual data by modifying the first half of the first or last byte (depending on whether you are using a sign-leading or sign-trailing representation). Typically, '0xC' is used to represent a positive number, '0xD' is used to represent a negative number and '0xF' is used to represent an unsigned number.

**Note:** In general, any of '0xA', '0xC', '0xE' or '0xF' may be used to indicate a positive value and '0xB' or '0xD' used to indicate a negative value. The actual preferred representation is dependent upon the actual hardware architecture.

On ASCII machines there are a number of mechanisms for the internal representation of external decimal data. One representation ('Sign ASCII') employed by IBM's pSeries machines uses the normal ASCII codes ("0" [hex 30] to "9" [hex 39]) for the first/last digit of both unsigned and positive numbers, and the characters "p" [hex 70] to "y" [hex 79] for negative numbers.

An alternative method (Sign EBCDIC Custom) is used on some other ASCII based machines. This uses the same characters as an EBCDIC based machine, even though the actual internal hexadecimal representation for them are different. Using this technique the character string for both EBCDIC and ASCII platforms is identical. You could potentially receive a message from an EBCDIC platform (created from a COBOL copy book that contains such entries as PIC XXX and PIC S999) and convert the whole message to ASCII or the other way around. The character string representing the external decimal field in the message (after the ASCII/EBCDIC conversion) maps to the code point which represents the correct sign for the decimal. You should note that there is a limitation with this method. Curly brace characters are variant (i.e. they have different code points in different EBCDIC codepages). This mechanism works only for those EBCDIC codepages where the curly brace characters '{' and '}' (which are used to represent signed 0) have exactly the code points x'C0' and x'D0'. For example, it will work for code page 500 and not for code page 871 where the curly braces have code points X'8E' X'9C.

In an ASCII environment (determined by the CCSID property at runtime), the default for both input and output is the 'Sign ASCII' representation. It is possible to specify the applicable representation in the CWF physical layer for local attributes and local elements of types decimal, float, and integer.

**Note:** This is only appropriate for those elements/attributes which have an external decimal physical representation and which have an embedded ('Leading' or 'Trailing') sign (determined by the 'Sign Orientation' property).

The table below show the internal representation (both character and actual hexadecimal value) of the first or last digit for external decimal numbers with an included (embedded) leading or trailing sign respectively. (Note: the table does not specify the representation for unsigned values which are 0x30-0x39 for ASCII and 0xF0-0xF9 for EBCDIC)



	Positively signed values			Negatively signed values		
	ASCII environment		EBCDIC environment	ASCII environment		EBCDIC environment
Digit	Sign ASCII	Sign EBCDIC Custom		Sign ASCII	Sign EBCDIC Custom	
0	0(30)	{(7B)	{(C0)	p(70)	}(7D)	}(D0)
1	1(31)	A(41)	A(C1)	q(71)	J(4A)	J(D1)
2	2(32)	B(42)	B(C2)	r(72)	K(4B)	K(D2)
3	3(33)	C(43)	C(C3)	s(73)	L(4C)	L(D3)
4	4(34)	D(44)	D(C4)	t(74)	M(4D)	M(D4)
5	5(35)	E(45)	E(C5)	u(75)	N(4E)	N(D5)
6	6(36)	F(46)	F(C6)	v(76)	O(4F)	O(D6)
7	7(37)	G(47)	G(C7)	w(77)	P(50)	P(D7)
8	8(38)	H(48)	H(C8)	x(78)	Q(51)	Q(D8)
9	9(39)	I(49)	I(C9)	y(79)	R(52)	R(D9)

This next table gives some examples for a range of simple numbers that would be representative of what could be transmitted or received using these approaches.

	Sign leading			Sign trailing		
	ASCII Environment		EBCDIC Environment	ASCII Environment		EBCDIC Environment
Decimal value	Sign ASCII	Sign EBCDIC Custom		Sign ASCII	Sign EBCDIC Custom	
1234	31 32 33 34 "1234"	31 32 33 34 "1234"	F1 F2 F3 F4 "1234"	31 32 33 34 "1234"	31 32 33 34 "1234"	F1 F2 F3 F4 "1234"
+1234	31 32 33 34 "1234"	41 32 33 34 "A234"	C1 F2 F3 F4 "A234"	31 32 33 34 "1234"	31 32 33 44 "123D"	F1 F2 F3 C4 "123D"
-1234	71 32 33 34 "q234"	4A 32 33 34 "J234"	D1 F2 F3 F4 "J234"	31 32 33 74 "123t"	31 32 33 4D "123M"	F1 F2 F3 D4 "123M"
7890	37 38 39 30 "7890"	37 38 39 30 "7890"	F7 F8 F9 F0 "7890"	37 38 39 30 "7890"	37 38 39 30 "7890"	F7 F8 F9 F0 "7890"
+7890	37 38 39 30 "7890"	47 38 39 30 "G890"	C7 F8 F9 F0 "G890"	37 38 39 30 "7890"	37 38 39 7B "789{"	F7 F8 F9 C0 "789{"
-7890	77 38 39 30 "w890"	50 38 39 30 "P890"	D7 F8 F9 F0 "P890"	37 38 39 70 "789p"	37 38 39 7D "789}"	F7 F8 F9 D0 "789}"

## Importing from WSDL: generated objects and restrictions

This describes the objects that are generated when a WSDL definition is imported and some restrictions that you need to be aware of.

### Generated objects

#### Files copied by command line import

The `mqsicreatemsgdefsfromwsdl` command copies the WSDL files it needs into the workspace before the import process is run. These are the top level WSDL file and any imports resolved via a relative location. The files are copied under the specified message set in to a folder called `importFiles`.

## Report File

The WSDL importer appends the result of the import operation to a report file, listing any errors that occurred during the process. The file name of the report file is <message set>.wsdl.report.txt.

## SOAP Message Definitions

The required SOAP mxsds are added to the message set. Currently the SOAP 1.1 definitions are always imported. There are several reasons for this:

- SOAP 1.1 is more widely used than SOAP 1.2.
- There is no standard SOAP 1.2 binding for WSDL 1.1. This will mean that the WSDL importer cannot reliably determine that SOAP 1.2 is required.
- It is not possible to import both SOAP 1.1 and SOAP 1.2 definitions because they use the same message name (for example, Envelope).

If you need to parse SOAP 1.2 instance documents you should manually remove the SOAP 1.1 definitions and import the SOAP 1.2 definitions via the Message Definition File wizard, selecting **IBM supplied message**.

If your message set has TDS or CWF layers you may find that you get a number of warnings against the imported SOAP definitions. Most of these can be ignored, but you may need to take account of the allowed values for boolean attributes. In SOAP 1.1 a boolean has the value 1 or 0, whilst in SOAP 1.2 the values are true and false. The XML representation of boolean values for a message set is specified in the physical properties for the XML physical format and may need to be set accordingly.

## Message Definition Files

Other message definition file names are created as <input file name>.mxsd and their content depends on the WSDL style.

### document-style

WSDL message parts for style="document" (which includes all SOAP header, fault and headerfault parts) refer to an element defined in XML Schema. This element is imported as a global element and broker message in the mxsd.

The xsi:type Output Policy on the message is set to "Never".

### rpc-style

WSDL message parts for style="rpc" (and exclusively those allocated to the SOAP body) refer to a type defined in XML Schema. In this case, input and output messages are created as follows:

	An input message	An output message
Derived From	wSDL:input child (if any) of WSDL operation, and the WSDL message and parts which it identifies	wSDL:output child (if any) of WSDL operation, and the WSDL message and parts which it identifies
Name of Element	value of the name attribute on the WSDL operation element	value of the name attribute on the WSDL operation element suffixed by "Response"
Namespace of Element	value of the namespace attribute on the corresponding soap:body element	value of the namespace attribute on the corresponding soap:body element

Each message is of local complex type, being a sequence of elements. The name of each element is the value of the name attribute on the WSDL parts of the message identified by the input or output element respectively. These elements have no namespace (the underlying schema representation has form="unqualified") and are locally scoped to avoid name clashes. The type of these local elements is the XML Schema type referred to by the type attribute of the corresponding part element. (The type will be global in the WSDL schema.)

If the soap:body was defined with use="encoded" in the WSDL definition then the message definition includes a reference to the attribute group encodingStyle in the SOAP-ENV namespace and the xsi:type Output Policy on the message is set to "Follow SOAP encoding rules". Otherwise the xsi:type Output Policy on the message is set to "Never".

## Restrictions

This describes some of the restrictions related to importing WSDL definitions. These are where the WSDL definitions are not WS-I compliant.

### SOAP Arrays

A WSDL 1.1 definition may define a SOAP Array (only applicable to the WSDL rpc-encoded style, and not WS-I compliant):

```
<xsd:complexType name="t">
  <xsd:complexContent>
    <xsd:restriction base="SOAP-ENC:Array">
      <xsd:sequence>
        <xsd:element name="item" type="string" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="xsd:string[]"/>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

Some uses of SOAP array syntax are not fully supported. Although a useful tree is created during parsing (and can be serialized when writing), note that:

- the model does not take any account of the SOAP-ENC:arrayType attribute.
- the model for partially transmitted arrays does not take account of the SOAP-ENC:offset attribute.  
For example, the first element of an array specified with offset[2] would be accessed in ESQL not as InputRoot.MRM.array.item[3], but as InputRoot.MRM.array.item[1].
- the model for multi-dimensional arrays flattens the representation into a single dimension. For example, a 2 dimensional array is accessed in ESQL not as InputRoot.MRM.array.item[x][y], but as InputRoot.MRM.array.item[i] where the index i would have to be calculated appropriately.

### Anonymous Elements

The WSDL excerpt above describes a SOAP instance document of the following form:

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[3]">
  <item xsi:type="xsd:string">A general text string</item>
  <item xsi:type="xsd:token">A restriction of the string type</item>
  <item xsi:type="xsd:Name">ARestrictionOfTheTokenType</item>
</SOAP-ENC:Array>
```

The broker model handles this as expected, but in SOAP encoding, array elements are also allowed to use the type-elements from the SOAP encoding namespace. This means an application using the same WSDL definition might create an instance document of the following form:

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[3]">
  <SOAP-ENC:string>A general text string</SOAP-ENC:string>
  <SOAP-ENC:token>A restriction of the string type</SOAP-ENC:token>
  <SOAP-ENC:Name>ARestrictionOfTheTokenType</ SOAP-ENC:Name>
</SOAP-ENC:Array>
```

The broker model created by importing the WSDL would have to be manually edited to handle this case, unless it is acceptable to have the parser treat it as a self-defined element.

## Importing from XML Schema: unsupported features

### Message sets with namespace support

- Constructs accepted but not supported when importing from an XML schema.

When importing an XML Schema into a message set that supports namespaces, the constructs below are accepted, but cause an error message to be displayed in the task list as they are not fully supported.

The following is a list of these schema declarations:

- Redefine

The following is an example of each of these constructs:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.ibm.com" xmlns:ibm="http://www.ibm.com">

  <!-- Unsupported feature: redefine -->
  <redefine schemaLocation="test.xsd"/>

</schema>
```

- Constructs accepted and ignored when importing from an XML schema.

When importing an XML Schema into a message set that supports namespaces, the constructs below are accepted, but will be ignored and will not be deployed to the broker. When the user tries to import a schema containing one or more of these constructs, a warning is issued stating that they will be ignored. You will be able to delete these constructs, but there are no properties that can be configured for them.

If you extract the logical model for the message (for example using the Schema Generation facility) these constructs are ignored and are not included in the output schema.

The following is a list of these schema declarations:

- Unique
- Key
- Keyref

The following is an example of each of these constructs:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.ibm.com" xmlns:ibm="http://www.ibm.com">

  <!-- Accepted feature: unique -->
  <element name="element1" type="string">
    <unique name="unique1">
      <selector xpath="path1"></selector>
      <field xpath="path1"></field>
```

```

    </unique>
  </element>

  <!-- Accepted feature: key -->
  <element name="element2" type="string">
    <key name="key1">
      <selector xpath="path1"></selector>
      <field xpath="path1"></field>
    </key>
  </element>

  <!-- Accepted feature: keyref -->
  <element name="element3" type="string">
    <keyref name="keyref1" refer="ibm:key1">
      <selector xpath="path1"></selector>
      <field xpath="path1"></field>
    </keyref>
  </element>

</schema>

```

- Where the Schema Location generated for some include statements could be incorrect.

The problem occurs when an XML schema that has a target namespace, includes another XML schema that has no target namespace. The Schema Location created in the message model for the include just specifies the filename. This does not take into account that the including and included message definition files are in different directories within the message model.

You can fix problem from the workbench by selecting the **Properties** tab of the including message definition file in the Message Definition Editor. Delete the include entry with the incorrect Schema Location then add another include entry, selecting the correct file from the file menu.

## Message sets without namespace support

- When importing an XML Schema into a message set that does not support namespaces, the following constructs can be accepted, rejected, or modified for reasons of compatibility with Version 2.1:
  - Redefine
  - Import
  - List
  - Union
  - Abstract complex type
  - Abstract element

The user has the choice to accept, reject or modify these constructs based on the preferences set by the user in the XML Schema Importer preference page (or the mqsicreatemsgdefs import options file).

If the user specifies "reject", when the construct is encountered, the import stops and an error is reported.

If the user specifies "modify" (the default), when the construct is encountered, the importer modifies the resulting message definition file as follows:

- Redefine statements are ignored and not imported.
- Import statements are replaced with include statements.
- Any list/union types are modified to be a restriction with a base type xsd:string.
- Abstract complex types are changed into non-abstract complex types.
- Abstract elements are changed into non abstract elements.

Note that these modifications are the equivalent modifications that were performed by the Version 2.1 XML Schema importer command, for compatibility. If the user specifies "accept", when the construct is encountered, the importer imports the construct as it is. This may result in task list errors depending on the construct, as described above for message sets with namespace support.

The following is an example of each of these constructs:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.ibm.com" xmlns:ibm="http://www.ibm.com">

  <!-- Supported feature: redefine -->
  <redefine schemaLocation="test.xsd"/>

  <!-- Supported feature: import -->
  <import schemaLocation="test1.xsd" namespace="http://www.ibm1.com"/>

  <!-- Supported feature: list -->
  <simpleType name="type1">
    <list itemType="string" />
  </simpleType>

  <!-- Supported feature: union -->
  <simpleType name="type2">
    <union memberTypes="string" />
  </simpleType>

  <!-- Supported feature: abstract complex type -->
  <complexType name="type3" abstract="true"></complexType>

  <!-- Supported feature: abstract element -->
  <element name="element" type="string" abstract="true"></element>

</schema>
```

- Constructs accepted and ignored when importing from an XML schema. The list of constructs and the action taken is the same as for a message set with namespace support, as described above.
- Target namespaces not qualified with a prefix. When importing an XML schema into a message set that does not support namespaces, you cannot import a schema document that has a target namespace that is not qualified with a prefix. For example:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.ibm.com" xmlns="http://www.ibm.com">
</xsd:schema>
```

## Importing from XML DTD: unsupported features

The same limitations that apply to importing an XML Schema also apply to importing an XML DTD. The DTD importer converts XML DTD to an XML schema and then the schema is imported. If the generated XML Schema happens to have any of the XML Schema limitations then the behavior is as described in "Importing from XML Schema: unsupported features" on page 732.

---

## Message model wizards

This provides a top level to the additional reference information for some of the more complex message modeling wizards.

It is the general aim of a wizard to simplify a repetitive task. As software has grown more complex, they are now used to step you through complex tasks or procedures to ensure that all the fields are correctly filled in as well as ensuring that each of the necessary steps are completed in the correct order. The aim of this section is to provide some additional reference material for those wizards where you could require additional assistance for certain fields.

The wizards that are documented will each have their own section and each panel in the wizard will have their own topic. The panels will be listed in the order they appear in the wizard and the fields on each of the panels will appear in the topic in the same order as the panel. These topics are specifically limited in scope to providing additional information on these fields and panels and further information should be sought from associated topics.

The following wizards have additional information:

- “New message definition file wizard”
- “New WSDL definition wizard” on page 764

**Related tasks**

“Developing message models” on page 3

**Related reference**

“Message model reference information” on page 145

## **New message definition file wizard**

This provides additional reference information in relation to the New message definition file wizard.

The following links provide further information in relation to the panels and fields that form the New message definition file wizard. The New message definition file wizard is used to create message definition files using a variety of methods. Depending on the selection that you make, you will be routed through the correct sequence of panels to create the message definition file from the source that you have requested. Some panels will only appear if certain conditions are met. These are marked as (optional).

The following are the methods that you can use the New message definition file wizard to create a message definition file:

- Create a new message definition file
- XML Schema file
- IBM supplied message
- XML DTD
- C header file
- COBOL file
- WSDL file

### **New message definition file wizard: Create a new message definition file**

New message definition file wizard, create a new message definition file reference material.

## Create a new message definition file

When you are using the option Create a new message definition file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select the target message set
- Specify the namespace

### Create a new message definition file: Select the message definition source:

New message definition file wizard, create a new message definition file option, message definition file source reference material.

## Create a new message definition file

When you are using the option Create a new message definition file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select the target message set
- Specify the namespace

## Panel properties

### Create a new message definition file

Select this option if you want to create the basic message definition file and define the content using the message definition editor.

### XML Schema file

Select this option if you want to create the message definition file by importing an XML Schema.

### IBM supplied message

Select this option if you want to create the message definition file by importing one of the IBM supplied messages.

### XML DTD file

Select this option if you want to create the message definition file by importing an XML DTD.

### C header file

Select this option if you want to create the message definition file by importing a C header.

### COBOL file

Select this option if you want to create the message definition file by importing a COBOL copybook.

### WSDL file

Select this option if you want to create the message definition file by importing a WSDL definition.

### Create a new message definition file: Select the target message set:

New message definition file wizard, create a new message definition file option, select the target message set reference material.



## Create a new message definition file

When you are using the option Create a new message definition file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select the target message set
- Specify the namespace

### Panel properties

#### Message sets

This field lists the message set projects that are available in your work space. By clicking on the '+' symbol to the left of the project, these will expand to list the message sets that are available for you to select. Subject to how you started the New message definition file wizard, a message set could be preselected for you. This does not prevent you from selecting an alternate should you choose to do so.

#### File name

Specify the name of the message definition file that you are creating.

#### Create a new message definition file: Specify the namespace:

New message definition file wizard, create a new message definition file option, specify the namespace reference material.

## Create a new message definition file

When you are using the option Create a new message definition file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select the target message set
- Specify the namespace

### Panel properties

#### Schema for Schema settings

**Prefix** The specifies the namespace prefix to use for the namespace shown in the **Namespace** property.

#### Namespace

Specifies the namespace to be used.

#### Use target namespace

Selecting this option allows you to specify a target namespace for the message definition file. You can only choose a target namespace if namespaces have been enabled in the message set.

#### Target namespace settings

**Prefix** The specifies the namespace prefix to use for the namespace shown in the **Namespace** property.

#### Namespace

Specifies the namespace to be used.

## **New message definition file wizard: XML Schema file**

New message definition file wizard, XML Schema file reference material.

### **XML Schema file**

When you are using the option XML Schema file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a schema file
- Select target message set
- Select global elements
- Namespace and URI prefix pairs (optional)

#### **XML Schema file: Select the message definition source:**

New message definition file wizard, XML Schema file option, message definition file source reference material.

### **XML Schema file**

When you are using the option XML Schema file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a schema file
- Select target message set
- Select global elements
- Namespace and URI prefix pairs (optional)

## **Panel properties**

### **Create a new message definition file**

Select this option if you want to create the basic message definition file and define the content using the message definition editor.

### **XML Schema file**

Select this option if you want to create the message definition file by importing an XML Schema.

### **IBM supplied message**

Select this option if you want to create the message definition file by importing one of the IBM supplied messages.

### **XML DTD file**

Select this option if you want to create the message definition file by importing an XML DTD.

### **C header file**

Select this option if you want to create the message definition file by importing a C header.

### **COBOL file**

Select this option if you want to create the message definition file by importing a COBOL copybook.

### **WSDL file**

Select this option if you want to create the message definition file by importing a WSDL definition.

### **XML Schema file: Select a schema file:**

New message definition file wizard, XML Schema file option, select a schema file reference material.

### **XML Schema file**

When you are using the option XML Schema file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a schema file
- Select target message set
- Select global elements
- Namespace and URI prefix pairs (optional)

### **Panel properties**

#### **Workbench files**

The field displays the projects, folders and XML Schema files that are available in the workbench. Use this field to select the schema file you are creating the message definition file from.

If the schema file you are looking to create the message definition file from is not listed, it is possible that it has not been imported into the workspace.

Subject to how you started the New message definition file wizard, the schema file could already be selected for you. This does not prevent you from selecting an alternate should you choose to do so.

### **XML Schema file: Select target message set:**

New message definition file wizard, XML Schema file option, select target message set reference material.

### **XML Schema file**

When you are using the option XML Schema file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a schema file
- Select target message set
- Select global elements
- Namespace and URI prefix pairs (optional)

### **Panel properties**

#### **Message sets**

This field lists the message set projects that are available in your work space. By clicking on the '+' symbol to the left of the project, these will expand to list the message sets that are available for you to select. Subject

to how you started the New message definition file wizard, a message set could be preselected for you. This does not prevent you from selecting an alternate should you choose to do so.

If you select a message set that does not support namespaces, a warning is shown and the imported XML Schema will be modified to remove namespaces.

**File name**

Specify the name of the message definition file that you are creating.

**Overwrite existing file contents without warning**

Select this option if you want to replace the contents of an existing message definition file and not be warned that this could occur. You are recommended to only select this if you are confident that replacing an existing message definition file will not cause you any problems.

**XML Schema file: Select global elements:**

New message definition file wizard, XML Schema file option, select global elements reference material.

**XML Schema file**

When you are using the option XML Schema file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a schema file
- Select target message set
- Select global elements
- Namespace and URI prefix pairs (optional)

**Panel properties****Global elements**

The field lists all the global elements based on global complex types that are in the validated schema file. In the displayed list of global elements, select the check boxes for the global elements from which you want to create messages.

**XML Schema file: Namespace and URI prefix pairs (optional):**

New message definition file wizard, XML Schema file option, namespace and URI prefix pairs reference material.

**XML Schema file**

When you are using the option XML Schema file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a schema file
- Select target message set
- Select global elements
- Namespace and URI prefix pairs (optional)

## Panel properties

### Namespace Prefix

Define the prefix that provides the namespace URI with its alias.

### Namespace URI

Provide a namespace URI that uniquely identifies the names space.

## New message definition file wizard: IBM supplied message

New message definition file wizard, IBM supplied message reference material.

## IBM supplied message

When you are using the option IBM supplied message in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select an IBM supplied message
- Select target message set
- Namespace and URI prefix pairs (optional)

### IBM supplied message: Select the message definition source:

New message definition file wizard, IBM supplied message option, message definition file source reference material.

## IBM supplied message

When you are using the option IBM supplied message in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select an IBM supplied message
- Select target message set
- Namespace and URI prefix pairs (optional)

## Panel properties

### Create a new message definition file

Select this option if you want to create the basic message definition file and define the content using the message definition editor.

### XML Schema file

Select this option if you want to create the message definition file by importing an XML Schema.

### IBM supplied message

Select this option if you want to create the message definition file by importing one of the IBM supplied messages.

### XML DTD file

Select this option if you want to create the message definition file by importing an XML DTD.

### C header file

Select this option if you want to create the message definition file by importing a C header.

### **COBOL file**

Select this option if you want to create the message definition file by importing a COBOL copybook.

### **WSDL file**

Select this option if you want to create the message definition file by importing a WSDL definition.

### **IBM supplied message: Select an IBM supplied message:**

New message definition file wizard, IBM supplied message option, select an IBM supplied message reference material.

## **IBM supplied message**

When you are using the option IBM supplied message in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select an IBM supplied message
- Select target message set
- Namespace and URI prefix pairs (optional)

## **Panel Options**

### **IBM supplied messages**

This field is used to select the IBM supplied message that you will create the message definition file from.

*IBM supplied messages that you can import:*

Description of the IBM supplied messages that you can import to create a new message definition file.

If the message is to be used with an XML parser, the following points apply:

- If the message set to which you are adding the new message definition file does have an XML wire format layer, but no namespace support, the imported IBM supplied message is modified to remove namespaces. It is therefore recommended that you enable namespace support before importing an IBM supplied message.
- If the message set to which you are adding the new message definition file does not have an XML wire format layer, but does have namespace support, only the logical information appears in the model. The IBM supplied message is not modified to remove namespaces. You can add the physical layer to the message set before or after importing the IBM supplied message but it is recommended that you add the physical layer before importing an IBM supplied message.
- If the message set to which you are adding the new message definition file does not have an XML wire format layer, and does not have namespace support, only the logical information appears in the model and the imported IBM supplied message is modified to remove namespaces.

The IBM supplied messages that you can import are:

### **SOAP message definitions**

These model the SOAP-defined portions of SOAP XML messages and are best used with the MRM, XMLNSC, or XMLNS parsers. The 'soapenv'

definitions model the SOAP envelope structure used to wrapper the user-defined body of a SOAP message. The 'soapenc' definitions model certain structures for use in "rpc/encoded" style SOAP messages. Separate definitions are supplied for versions 1.1 and 1.2 of the SOAP standard.

### **Multipart MIME message definitions**

These model the MIME-defined portions of multipart MIME messages and should be used with the message broker's MIME parser. Use the 'mime\_multipart' definition for normal multipart MIME messages such as SOAP with Attachments or RosettaNet. Use the 'mime\_nested\_multipart' definition for multipart MIME messages where the individual parts can themselves be multipart MIME, an example being S/MIME.

### **SAP IDoc message definition**

This models the SAP-defined portion of an IDoc which precedes the user-defined content. You must use it with the message broker's IDOC parser.

### **TimeoutRequest message definition**

This models the TimeoutRequest message used in conjunction with the message broker TimeoutControl and TimeoutNotification nodes. You can use it with any parser.

### **IBM supplied message: Select target message set:**

New message definition file wizard, IBM supplied message option, select target message set reference material.

## **IBM supplied message**

When you are using the option IBM supplied message in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select an IBM supplied message
- Select target message set
- Namespace and URI prefix pairs (optional)

## **Panel properties**

### **Message sets**

This field lists the message set projects that are available in your work space. By clicking on the '+' symbol to the left of the project, these will expand to list the message sets that are available for you to select. Subject to how you started the New message definition file wizard, a message set could be preselected for you. This does not prevent you from selecting an alternate should you choose to do so.

If you select a message set that does not support namespaces, a warning is shown and the imported IBM supplied message will be modified to remove namespaces.

### **File name**

Specify the name of the message definition file that you are creating.

### **Overwrite existing file contents without warning**

Select this option if you want to replace the contents of an existing message definition file and not be warned that this could occur. You are

recommended to only select this if you are confident that replacing an existing message definition file will not cause you any problems.

#### **IBM supplied message: Select global elements:**

New message definition file wizard, IBM supplied message option, select global elements reference material.

#### **IBM supplied message**

When you are using the option IBM supplied message in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select an IBM supplied message
- Select target message set
- Namespace and URI prefix pairs (optional)

#### **Panel properties**

##### **Global elements**

The field lists all the global elements in the IBM supplied message. In the displayed list of global elements, select the check boxes for the global elements from which you want to create messages.

##### **Namespace and URI prefix pairs (optional):**

New message definition file wizard, IBM supplied message option, namespace and URI prefix pairs reference material.

#### **IBM supplied message**

When you are using the option IBM supplied message in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select an IBM supplied message
- Select target message set
- Namespace and URI prefix pairs (optional)

#### **Panel properties**

##### **Namespace Prefix**

Define the prefix that provides the namespace URI with its alias.

##### **Namespace URI**

Provide a namespace URI that uniquely identifies the names space.

#### **New message definition file wizard: XML DTD file**

New message definition file wizard, XML DTD file reference material.

#### **XML DTD file**

When you are using the option XML DTD file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:



- Select the message definition source
- Select a DTD file
- Select target message set
- Select global elements

**XML DTD file: Select the message definition source:**

New message definition file wizard, XML DTD file option, message definition file source reference material.

**XML DTD file**

When you are using the option XML DTD file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a DTD file
- Select target message set
- Select global elements

**Panel properties**

**Create a new message definition file**

Select this option if you want to create the basic message definition file and define the content using the message definition editor.

**XML Schema file**

Select this option if you want to create the message definition file by importing an XML Schema.

**IBM supplied message**

Select this option if you want to create the message definition file by importing one of the IBM supplied messages.

**XML DTD file**

Select this option if you want to create the message definition file by importing an XML DTD.

**C header file**

Select this option if you want to create the message definition file by importing a C header.

**COBOL file**

Select this option if you want to create the message definition file by importing a COBOL copybook.

**WSDL file**

Select this option if you want to create the message definition file by importing a WSDL definition.

**XML DTD file: Select a DTD file:**

New message definition file wizard, XML DTD file option, select a DTD file reference material.

## XML DTD file

When you are using the option XML DTD file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a DTD file
- Select target message set
- Select global elements

### Panel properties

#### Workbench files

The field displays the projects, folders and XML DTD files that are available in the workbench. Use this field to select the XML DTD you are creating the message definition file from.

If the DTD file you are looking to create the message definition file from is not listed, it is possible that it has not been imported into the workspace.

Subject to how you started the New message definition file wizard, the DTD file could already be selected for you. This does not prevent you from selecting an alternate should you choose to do so.

#### XML DTD file: Select target message set:

New message definition file wizard, XML DTD file option, select a target message set reference material.

## XML DTD file

When you are using the option XML DTD file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a DTD file
- Select target message set
- Select global elements

### Panel properties

#### Message sets

This field lists the message set projects that are available in your work space. By clicking on the '+' symbol to the left of the project, these will expand to list the message sets that are available for you to select. Subject to how you started the New message definition file wizard, a message set could be preselected for you. This does not prevent you from selecting an alternate should you choose to do so.

If you select a message set that does not support namespaces, a warning is shown and the imported XML DTD will be modified to remove namespaces.

#### File name

Specify the name of the message definition file that you are creating.

#### Overwrite existing file contents without warning

Select this option if you want to replace the contents of an existing

message definition file and not be warned that this could occur. You are recommended to only select this if you are confident that replacing an existing message definition file will not cause you any problems.

#### **XML DTD file: Select global elements:**

New message definition file wizard, XML DTD file option, select global elements reference material.

#### **XML DTD file**

When you are using the option XML DTD file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a DTD file
- Select target message set
- Select global elements

#### **Panel properties**

##### **Global elements**

The field lists all the global elements based on global complex types that are in the validated XML DTD file. In the displayed list of global elements, select the check boxes for the global elements from which you want to create messages.

#### **New message definition file wizard: C header file**

New message definition file wizard, C header file reference material.

#### **C header file**

When you are using the option C header file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a C header file
- Select target message set
- Select the structures to import
- Select the physical properties

#### **C header file: Select the message definition source:**

New message definition file wizard, C header file option, message definition file source reference material.

#### **C header file**

When you are using the option C header file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a C header file
- Select target message set

- Select the structures to import
- Select the physical properties

## Panel properties

### Create a new message definition file

Select this option if you want to create the basic message definition file and define the content using the message definition editor.

### XML Schema file

Select this option if you want to create the message definition file by importing an XML Schema.

### IBM supplied message

Select this option if you want to create the message definition file by importing one of the IBM supplied messages.

### XML DTD file

Select this option if you want to create the message definition file by importing an XML DTD.

### C header file

Select this option if you want to create the message definition file by importing a C header.

### COBOL file

Select this option if you want to create the message definition file by importing a COBOL copybook.

### WSDL file

Select this option if you want to create the message definition file by importing a WSDL definition.

### C header file: Select a C header file:

New message definition file wizard, C header file option, select a C header file reference material.

## C header file

When you are using the option C header file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a C header file
- Select target message set
- Select the structures to import
- Select the physical properties

## Panel properties

### Workbench Files

The field displays the projects, folders and C header files that are available in the workbench. Use this field to select the C header file you are creating the message definition file from.

If the C header file you are looking to create the message definition file from is not listed, it is possible that it has not been imported into the workspace.

Subject to how you started the New message definition file wizard, the C header file could already be selected for you. This does not prevent you from selecting an alternate should you choose to do so.

### **Include Path**

The include path is used to define any paths that are required to ensure that the C data structure is imported correctly.

Use the **Add** and **Remove** buttons to ensure all necessary paths are correctly specified.

### **C header file: Select target message set:**

New message definition file wizard, C header file option, select target message set reference material.

### **C header file**

When you are using the option C header file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a C header file
- Select target message set
- Select the structures to import
- Select the physical properties

### **Panel properties**

#### **Message sets**

This field lists the message set projects that are available in your work space. By clicking on the '+' symbol to the left of the project, these will expand to list the message sets that are available for you to select. Subject to how you started the New message definition file wizard, a message set could be preselected for you. This does not prevent you from selecting an alternate should you choose to do so.

#### **Message definition file name**

Specify the name of the message definition file that you are creating.

#### **Overwrite existing file contents without warning**

Select this option if you want to replace the contents of an existing message definition file and not be warned that this could occur. You are recommended to only select this if you are confident that replacing an existing message definition file will not cause you any problems.

#### **Preserve case in variable names**

By selecting this option, the importer will preserve the case that the variable names in the C data structure have been written in. By deselecting this option, all variable names will be in lowercase.

The default is for this to be selected.

#### **Use target namespace**

Selecting this option allows you to specify a target namespace for the message definition file. You can only choose a target namespace if namespaces have been enabled in the message set.

If you select this option, specify the namespace to be used in the field that has been enabled.

The namespace prefix created in the message definition file will be set to tns. You can edit the Namespace Settings on the XML physical format layer at the message set level to control prefixes written in XML instance messages.

### **C header file: Select the structures to import:**

New message definition file wizard, C header file option, select target message set reference material.

## **C header file**

When you are using the option C header file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a C header file
- Select target message set
- Select the structures to import
- Select the physical properties

## **Panel properties**

### **Source structures**

This field list the structures that are in the C header file that are available to import into the message definition file that you are creating. Use this field to select the structures you want to import and use the > button to move them into the **Imported structures** field.

Use the >> button to move all the structures listed into the Imported structures field.

### **Imported structures**

This fields lists all the structures that you have selected from the **Source structures** field.

Select the check boxes for those structures in the list to create as messages in the message definition file that you are creating.

Use the < to remove selected structures from the list or the << button to remove all structures from the list.

### **Prefix for imported structure names (optional)**

A prefix can be added to the names of all the imported global complex types and groups created during the import. Adding a prefix can help to avoid the possible duplication of names.

### **C header file: Select the physical properties:**

New message definition file wizard, C header file option, select target message set reference material.

## C header file

When you are using the option C header file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a C header file
- Select target message set
- Select the structures to import
- Select the physical properties

## Panel properties

### Source platform

This property sets the defaults for Codepage, Floating point format, and Byte order.

- (Unspecified)
- Msvc
- icc
- AIXgcc
- AIXxlC
- OS390

### Codepage

This shows the default for what has been set using the Source platform property. This should not be adjusted.

- ISO8859-1
- Cp037
- Cp1252

### Floating point format

This shows the default for what has been set using the Source platform property. This should not be adjusted.

- IEEE Extended INTEL
- IEEE Extended AIX
- IEEE Extended OS/390
- IEEE Non-Extended
- IBM 390 Hexadecimal

## Storage layout

### Address size

Specify the native address size of the hardware that the C header was designed for. This affects:

- the CWF length of an imported pointer (4|8 bytes)
- the CWF length of a 'long' (4|8 bytes)

Select the size matching that of the program producing or consuming CWF messages.

- 32 bit - the default
- 64 bit

### Size of long double

This shows the default for what has been set using the Source platform property. This should not be adjusted.

- 64 bit - the default
- 128 bit

### Byte order

- Little Endian - the default
- Big Endian

### Pack level

This affects the CWF Byte Alignment and Leading and Trailing Skip Count of assigned elements. A C importer would naturally align a variable on a boundary equal to its own size (a 4 byte integer aligns on a 4 byte boundary). The compiler would add slack space between variables in a structure to allow this. The Pack setting gives a maximum alignment value for a variable. If your messages contain slack space, select the pack level matching that of the program producing or consuming CWF messages. Select '1' if no slack space is desired. For example:

```
/* Test data alignment
 * Struct containing an int (naturally aligns on word boundary)
 *
 * +-----+
 * | a,b,c = a byte of the element
 * | _ = a byte of padding
 * +-----+
 *
 *          111
 * pack    | 1234 5678 9012 | Total structure size
 * +-----+-----+-----+
 * 1-byte  | abbb bc       | 6 bytes
 * 2-byte  | a_bb bbc_    | 8 bytes
 * 4-byte  | a__ bbbb c__ | 12 bytes
 * 8-byte  | a___ bbbb c__ | 12 bytes
 * +-----+-----+-----+
 *
 * *****/
struct _align01 {
    char align_01_a;
    int align_01_b;
    char align_01_c;
};
```

- 1
- 2
- 3
- 8 - the default
- 16

### Enum size

Size (in bytes) of an enum. This relates directly to the CWF Length Count. Setting Compact will set the smallest size that contains the largest value in the enumeration. Select the size matching that of the program producing or consuming CWF messages

- 1
- 2
- 4
- Compact - the default

### String encoding



- Null terminated. When Null terminated is selected, a C char array of size n is imported as an element of type xsd:string with a maxLength facet of n-1. If a CWF physical format is defined for the message set, the CWF physical type of the element is Null Terminated String.
- Fixed length. When Fixed length is selected (the default), a C char array of size n is imported as an element of type xsd:string with a maxLength facet of n. If a CWF physical format is defined for the message set, the CWF physical type of the element is Fixed Length String with a Length Count of n.

This option only applies to character arrays that are currently imported as xsd:string. For example, it does not include "unsigned char[2]" or "signed char[2]" both of which are imported as restrictions of xsd:integer.

### **Padding Char for String**

Enter the padding character that is to be used when importing the C header file. Either select from one of the predefined entries in the listed, or specify one by typing it into the field. The value must resolve to a single character. Set this character in one of the following ways:

- Select NUL (the default), SPACE, or '0' from the drop-down list.
- Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.
- Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.
- Enter a decimal character code in the form YY where YY is a decimal value.
- Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal.

This option is greyed out if the String Encoding option is other than Fixed length.

### **New message definition file wizard: COBOL file**

New message definition file wizard, COBOL file reference material.

#### **COBOL file**

When you are using the option COBOL file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a COBOL file
- Select target message set
- Select the structures to import
- Select the physical properties

#### **COBOL file: Select the message definition source:**

New message definition file wizard, COBOL file option, message definition file source reference material.

## COBOL file

When you are using the option COBOL file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a COBOL file
- Select target message set
- Select the structures to import
- Select the physical properties

## Panel properties

### Create a new message definition file

Select this option if you want to create the basic message definition file and define the content using the message definition editor.

### XML Schema file

Select this option if you want to create the message definition file by importing an XML Schema.

### IBM supplied message

Select this option if you want to create the message definition file by importing one of the IBM supplied messages.

### XML DTD file

Select this option if you want to create the message definition file by importing an XML DTD.

### C header file

Select this option if you want to create the message definition file by importing a C header.

### COBOL file

Select this option if you want to create the message definition file by importing a COBOL copybook.

### WSDL file

Select this option if you want to create the message definition file by importing a WSDL definition.

### COBOL file: Select a COBOL file:

New message definition file wizard, COBOL file option, select a COBOL file reference material.

## COBOL file

When you are using the option COBOL file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a COBOL file
- Select target message set
- Select the structures to import
- Select the physical properties

## Panel properties

### Workbench files

The field displays the projects, folders and COBOL files that are available in the workbench. Use this field to select the COBOL file you are creating the message definition file from.

If the COBOL file you are looking to create the message definition file from is not listed, it is possible that it has not been imported into the workspace.

Subject to how you started the New message definition file wizard, the COBOL file could already be selected for you. This does not prevent you from selecting an alternate should you choose to do so.

### COBOL file: Select target message set:

New message definition file wizard, COBOL file option, select target message set reference material.

## COBOL file

When you are using the option COBOL file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a COBOL file
- Select target message set
- Select the structures to import
- Select the physical properties

## Panel properties

### Message sets

This field lists the message set projects that are available in your work space. By clicking on the '+' symbol to the left of the project, these will expand to list the message sets that are available for you to select. Subject to how you started the New message definition file wizard, a message set could be preselected for you. This does not prevent you from selecting an alternate should you choose to do so.

If you select a message set that does not have a Custom Wire Format layer defined, a warning is shown and only the logical information will appear in the message definition file.

### Message definition file name

Specify the name of the message definition file that you are creating.

### Overwrite existing file contents without warning

Select this option if you want to replace the contents of an existing message definition file and not be warned that this could occur. You are recommended to only select this if you are confident that replacing an existing message definition file will not cause you any problems.

### Preserve case in variable names

By selecting this option, the importer will preserve the case that the variable names in the COBOL data structure have been written in. By deselecting this option, all variable names will be in lowercase.

The default is for this to be selected.

### Use target namespace

Selecting this option allows you to specify a target namespace for the message definition file. You can only choose a target namespace if namespaces have been enabled in the message set.

If you select this option, specify the namespace to be used in the field that has been enabled.

The namespace prefix created in the message definition file will be set to tns. You can edit the Namespace Settings on the XML physical format layer at the message set level to control prefixes written in XML instance messages.

### COBOL file: Select the structures to import:

New message definition file wizard, COBOL file option, select the structures to import reference material.

### COBOL file

When you are using the option COBOL file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a COBOL file
- Select target message set
- Select the structures to import
- Select the physical properties

### Panel properties

#### Source structures

This field list the structures that are in the COBOL file that are available to import into the message definition file that you are creating. Use this field to select the structures you want to import and use the > button to move them into the **Imported structures** field.

Use the >> button to move all the structures listed into the Imported structures field.

#### Imported structures

This fields lists all the structures that you have selected from the **Source structures** field.

Select the check boxes for those structures in the list to create as messages in the message definition file that you are creating.

Use the < to remove selected structures from the list or the << button to remove all structures from the list.

#### Prefix for imported structure names (optional)

A prefix can be added to the names of all the imported global complex types and groups created during the import. Adding a prefix can help to avoid the possible duplication of names.

### COBOL file: Select the physical properties:

New message definition file wizard, COBOL file option, select the physical properties reference material.

## COBOL file

When you are using the option COBOL file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a COBOL file
- Select target message set
- Select the structures to import
- Select the physical properties

## Panel properties

### Compiler information

#### Source platform

This property sets the defaults for Codepage, Floating point format, and Byte order.

- Win32 - the default
- AIX
- z/OS

#### Codepage

This shows the default for what has been set using the Source platform property. This should not be adjusted.

- ISO8859\_1 - the default

#### Floating point format

This shows the default for what has been set using the Source platform property. This should not be adjusted.

- IEEE Non-Extended - the default
- IBM 390 Hexadecimal

## Storage layout

#### Byte order

This shows the default for what has been set using the Source platform property. This should not be adjusted.

- Little endian - the default
- Big endian

#### External decimal sign

This sets the "Sign EBCDIC Custom" check box for elements of "CWF Physical Type" that are set to "External Decimal" and with the "Sign Orientation" set to Leading or Trailing (an embedded sign representation). This would correspond to a COBOL data item such as "USAGE DISPLAY PIC S999 SIGN LEADING" or "USAGE DISPLAY PIC S999 SIGN TRAILING".

You would require "EBCDIC Custom" if instance messages in an ASCII codepage will use EBCDIC-style embedded signs. Otherwise, ASCII or EBCDIC will not set the "Sign EBCDIC Custom" check box for elements and at runtime, the type of embedded sign is determined from the codepage of the instance message.

- ASCII - the default

- EBCDIC
- EBCDIC Custom

## Compile options

### QUOTE

This affects the character produced by the COBOL keywords QUOTE and QUOTES and is used for setting INITIAL VALUE of a data-item, imported as a default value.

- DOUBLE - the default
- SINGLE

### TRUNC

Implements the effects of the COBOL compiler option TRUNC(STD|OPT|BIN).

For STD or OPT, COBOL BINARY data-items (including USAGE COMP, COMP-4, or BINARY) have maxInclusive and minInclusive facets created conforming to their PICTURE clause (for example, PIC S999 = min -999, max 999).

For BIN, BINARY data-items behave as if created with USAGE COMP-5. No facets are created, the logical type of each element created (for example, xsd:short) gives the implicit maximum and minimum values that may be handled.

- STD - the default
- OPT
- BIN

### NSYMBOL

This is the equivalent of the COBOL compiler option NSYMBOL(NATIONAL|DBCS), which selects whether PIC N data-items are national or dbc items.

- DBCS
- NATIONAL - the default

### Create default values from INITIAL VALUES

If you want to create default values from the initial values, select the **Create default values from INITIAL VALUES** check box.

### Create facets from level 88 VALUE clauses where possible

If you want to create minimum inclusive facets, maximum inclusive facets and enumeration facets from the imported COBOL Level 88 values on the simple type associated with the Schema xsd:element, select the **Create facets from level 88 value clauses where possible** check box.

This adds a **Value Constraints** node under **Logical properties** in the **Properties Hierarchy** and the values that you specify are subsequently used for validation during runtime.

**Note:** In cases where a simple type has both minimum and maximum inclusive facets set as well as enumeration facets, the enumeration facets are used in preference to the minimum and maximum inclusive facets.

### Create null values for all fields

If your COBOL level 01 data structure is being initialized to a particular

character (for example, SPACE, LOW-VALUE, or HIGH\_VALUE) in your COBOL application, consider checking the **Create null values for all fields** check box.

When the data structure is initialized in this way, the resultant message can give rise to parsing errors because some fields contain a value that is not valid for the data type of the field. Checking the box enables you to treat any fields that have been initialized in this manner as NULL values and will enable the parsing to succeed.

You specify the initialization character using the **Null character** control, the default value for which is the SPACE character. Setting these controls will result in all global elements having their Nillable property set, their Custom Wire Format Encoding Null property set to NullLiteralFill, and their Custom Wire Format Encoding Null Value property set to the specified character.

### **NULL character**

Set the **Null character** in one of the following ways:

- Select SPACE (the default), NUL, 0x00 or 0xFF from the drop-down list.
- Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character.
- Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.
- Enter a decimal character code in the form YY where YY is a decimal value.
- Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal

Under normal collation sequence rules, use 0x00 for low-values and 0xFF for high-values.

### **Padding character for strings**

Enter the padding character that is to be used when importing the C header file. Either select from one of the predefined entries in the listed, or specify one by typing it into the field. The value must resolve to a single character. Set this character in one of the following ways:

- Select NUL, SPACE (the default), or '0' from the drop-down list.
- Enter a character between quotes, for example "c" or 'c', where c is any alphanumeric character.
- Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value.
- Enter a decimal character code in the form YY where YY is a decimal value.
- Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal.

## **New message definition file wizard: WSDL file**

New WSDL definition wizard, message definition file source reference material.

### **WSDL file**

When you are using the option WSDL file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source

- Select a WSDL file
- Select target message set
- Select WSDL binding and WS-I validation
- Specify the namespace prefixes
- The following files will be generated

**WSDL file: Select the message definition source:**

New message definition file wizard, WSDL file option, message definition file source reference material.

**WSDL file**

When you are using the option WSDL file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a WSDL file
- Select target message set
- Select WSDL binding and WS-I validation
- Specify the namespace prefixes
- The following files will be generated

**Panel properties**

**Create a new message definition file**

Select this option if you want to create the basic message definition file and define the content using the message definition editor.

**XML Schema file**

Select this option if you want to create the message definition file by importing an XML Schema.

**IBM supplied message**

Select this option if you want to create the message definition file by importing one of the IBM supplied messages.

**XML DTD file**

Select this option if you want to create the message definition file by importing an XML DTD.

**C header file**

Select this option if you want to create the message definition file by importing a C header.

**COBOL file**

Select this option if you want to create the message definition file by importing a COBOL copybook.

**WSDL file**

Select this option if you want to create the message definition file by importing a WSDL definition.

**WSDL file: Select a WSDL file:**

New message definition file wizard, WSDL file option, select a WSDL file reference material.



## WSDL file

When you are using the option WSDL file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a WSDL file
- Select target message set
- Select WSDL binding and WS-I validation
- Specify the namespace prefixes
- The following files will be generated

## Panel properties

### WSDL file name

The field displays the projects, folders and WSDL files that are available in the workbench. Use this field to select the WSDL file you are creating the message definition file from.

If the WSDL file you are looking to create the message definition file from is not listed, it is possible that it has not been imported into the workspace.

Subject to how you started the New message definition file wizard, the WSDL file could already be selected for you. This does not prevent you from selecting an alternate should you choose to do so.

### WSDL file: Select target message set:

New message definition file wizard, WSDL file option, select target message set reference material.

## WSDL file

When you are using the option WSDL file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a WSDL file
- Select target message set
- Select WSDL binding and WS-I validation
- Specify the namespace prefixes
- The following files will be generated

## Panel properties

### Message sets

This field lists the message set projects that are available in your work space. By clicking on the '+' symbol to the left of the project, these will expand to list the message sets that are available for you to select. Subject to how you started the New message definition file wizard, a message set could be preselected for you. This does not prevent you from selecting an alternate should you choose to do so.

If you select a message set that does not support namespaces, a warning is shown as you are only able to import a WSDL file into a message set that is namespace enabled.

### **WSDL file: Select WSDL binding and WS-I validation:**

New message definition file wizard, WSDL file option, select WSDL binding and WS-I validation reference material.

### **WSDL file**

When you are using the option WSDL file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a WSDL file
- Select target message set
- Select WSDL binding and WS-I validation
- Specify the namespace prefixes
- The following files will be generated

### **Panel properties**

#### **Select the WSDL binding to import**

This lists the WSDL bindings that are available in the WSDL file that has been selected. Select the binding you want to create the message definition file from.

#### **WS-I validation failure action**

Prior to importing your WSDL service or definition file, it will be validated against the Web Services Interoperability Basic Profile. If a validation failure occurs, you have the following options to specify how the import should proceed:

- Ignore. Any warnings from the WS-I validator will be written to the log file as informational. The user is unaware of the messages unless they view the log file. The user will be able to continue with the process.
- Warn. Any warnings from the WS-I validator will be written to the log file as warnings. Any messages that generate a warning will be highlighted with yellow symbols on the next panel of the wizard and the user will be able to continue with the process.
- Fail. Any warnings from the WS-I validator will be written to the log file as errors. Any messages that generate an error will be highlighted with red symbols on the next panel of the wizard and the user will not be able to continue with the process until the errors in the original WSDL definition are corrected and it is revalidated.

### **WSDL file: Specify the namespace prefixes:**

New message definition file wizard, WSDL file option, specify the namespace prefixes reference material.

### **WSDL file**

When you are using the option WSDL file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a WSDL file

- Select target message set
- Select WSDL binding and WS-I validation
- Specify the namespace prefixes
- The following files will be generated

## Panel properties

### Namespace prefix

This column specifies the default namespace prefix for each of the namespaces that have been identified from the WSDL definition that you are importing. These will be used as the namespace prefix in the files that are generated. You can either accept the defaults or over-type them with a prefix you would prefer.

### Namespace

This column specifies the namespaces that have been identified from the WSDL definition that you are importing.

### WSDL file: The following files will be generated:

New message definition file wizard, WSDL file option, the following files will be generated reference material.

## WSDL file

When you are using the option WSDL file in the New message definition file wizard to create a message definition, the following are the list of panels you will be presented with:

- Select the message definition source
- Select a WSDL file
- Select target message set
- Select WSDL binding and WS-I validation
- Specify the namespace prefixes
- The following files will be generated

## Panel properties

### Filename

This column specifies the default filename for each of the files that will be generated when importing the WSDL definition. You can either accept the defaults or over-type them with the filename you would prefer.

The check box on the left of each row is used to define whether you want the process to overwrite an existing file of the same name in the specified location. If the box has a tick in it and is grayed, then no file of that name exists in the specified location and the file will be generated.

If a file exists in the specified location with that filename, the check box will not be selected and should you proceed, these files will not be generated unless you select the check box. The existing files will then be overwritten.

If you change the filename, the state of the check box depends on whether a file with that name exists in the specified location. For example:

- If the filename you choose does not exist in the specified location, the check box will be selected and grayed.

- If the filename did not exist and you change it to one that does exist, the check box will remain selected, but will no longer be grayed. This will allow you to deselect it if required

#### **Location**

This column specifies the location the related file will be written to.

## **New WSDL definition wizard**

This provides additional reference information in relation to the New WSDL definition wizard.

The following links provide further information in relation the panels and fields that form the New WSDL definition wizard. Some panels will only appear if certain conditions are met. These are marked as (optional).

### **New WSDL definition wizard**

List of panels:

- Important information (optional)
- Select a message set folder
- Select destination and format details
- Specify WSDL details
- Select the categories
- Select the bindings
- Specify service details for SOAP/HTTP (optional)
- Specify binding properties for SOAP/JMS (optional)
- Specify service properties for SOAP/JMS (optional)
- Summary of tasks

### **New WSDL definition wizard: Important information (optional)**

New WSDL definition wizard, important information reference material.

### **New WSDL definition wizard**

List of panels:

- Important information (optional)
- Select a message set folder
- Select destination and format details
- Specify WSDL details
- Select the categories
- Select the bindings
- Specify service details for SOAP/HTTP (optional)
- Specify binding properties for SOAP/JMS (optional)
- Specify service properties for SOAP/JMS (optional)
- Summary of tasks

### **Panel properties**

#### **Always show this screen**

The first screen displays some important information about the role of message categories in WSDL generation. If you don't want to see this message again, deselect the check box.

## **New WSDL definition wizard: Select a message set folder**

New WSDL definition wizard, select a message set folder reference material.

### **New WSDL definition wizard**

List of panels:

- Important information (optional)
- Select a message set folder
- Select destination and format details
- Specify WSDL details
- Select the categories
- Select the bindings
- Specify service details for SOAP/HTTP (optional)
- Specify binding properties for SOAP/JMS (optional)
- Specify service properties for SOAP/JMS (optional)
- Summary of tasks

### **Panel properties**

#### **Message set folders**

Select the message set from which you want to generate a WSDL definition.

## **New WSDL definition wizard: Select destination and format details**

New WSDL definition wizard, select destination and format details reference material.

### **New WSDL definition wizard**

List of panels:

- Important information (optional)
- Select a message set folder
- Select destination and format details
- Specify WSDL details
- Select the categories
- Select the bindings
- Specify service details for SOAP/HTTP (optional)
- Specify binding properties for SOAP/JMS (optional)
- Specify service properties for SOAP/JMS (optional)
- Summary of tasks

### **Panel properties**

#### **WSDL output folder**

Select the destination folder for the web service definition files that will be generated.

#### **File Format**

Select from:

- single. The WSDL document is written to a single file. This format is more widely understood by external applications and SOAP toolkits.

- multi. The WSDL document is split into multiple files. This format could offer better reuse of the component files.

**WSDL Version**

Select the required version of WSDL.

**SOAP Version**

Select the required version of SOAP

**New WSDL definition wizard: Specify WSDL details**

New WSDL definition wizard, specify WSDL details reference material.

**New WSDL definition wizard**

List of panels:

- Important information (optional)
- Select a message set folder
- Select destination and format details
- Specify WSDL details
- Select the categories
- Select the bindings
- Specify service details for SOAP/HTTP (optional)
- Specify binding properties for SOAP/JMS (optional)
- Specify service properties for SOAP/JMS (optional)
- Summary of tasks

**Panel properties**

**Style** The style determines the format of the runtime SOAP messages described by the generated WSDL. The choices are:

- rpc
- document

**WSDL Namespace**

This must be a valid URI and becomes the target namespace for the WSDL definitions. This value has no particular significance outside of the WSDL definition itself and does not correspond to the namespace of SOAP messages described by the generated WSDL. A default value of `http://tempuri.org/<message set name>` is set.

**RPC Namespace**

This field is only enabled if you selected the Style as rpc. It is the namespace for the immediate children of your SOAP body. The value must be a valid URI. A default value of `http://tempuri.org/<message set name>` is set.

**Definition Name**

This is used in deriving the names of the WSDL file or files to be created, The default value is the name of your message set.

**Documentation**

Optional: This text is included as documentation for the PortType element on the generated WSDL. It has no implications for the SOAP messages described by the generated WSDL.

**New WSDL definition wizard: Select the categories**

New WSDL definition wizard, select the categories reference material.

## New WSDL definition wizard

List of panels:

- Important information (optional)
- Select a message set folder
- Select destination and format details
- Specify WSDL details
- Select the categories
- Select the bindings
- Specify service details for SOAP/HTTP (optional)
- Specify binding properties for SOAP/JMS (optional)
- Specify service properties for SOAP/JMS (optional)
- Summary of tasks

### Panel properties

#### Create operations for message categories.

Select the message categories for which you want to create WSDL operations. The category name becomes the WSDL operation name and the category contents determine the operation definition.

**Note:** Only message categories with a Message Category Kind set to wsdl will be listed.

### New WSDL definition wizard: Select the bindings

New WSDL definition wizard, select the bindings reference material.

## New WSDL definition wizard

List of panels:

- Important information (optional)
- Select a message set folder
- Select destination and format details
- Specify WSDL details
- Select the categories
- Select the bindings
- Specify service details for SOAP/HTTP (optional)
- Specify binding properties for SOAP/JMS (optional)
- Specify service properties for SOAP/JMS (optional)
- Summary of tasks

### Panel properties

#### SOAP (over HTTP)

Select SOAP (over HTTP) if you intend to use the HTTP transport protocol to send your SOAP format messages.

#### SOAP (over JMS)

Select SOAP (over JMS) if you intend to use the JMS transport protocol to send your SOAP format messages.

## New WSDL definition wizard: Specify service details for SOAP/HTTP (optional)

New WSDL definition wizard, specify service details for SOAP/HTTP reference material.

### New WSDL definition wizard

List of panels:

- Important information (optional)
- Select a message set folder
- Select destination and format details
- Specify WSDL details
- Select the categories
- Select the bindings
- Specify service details for SOAP/HTTP (optional)
- Specify binding properties for SOAP/JMS (optional)
- Specify service properties for SOAP/JMS (optional)
- Summary of tasks

### Panel properties

#### SOAP Action

This defines the value for the HTTP SoapAction header. It is possible that an application will use the SoapAction as a mechanism for relating a SOAP message to an implementation method. This is often true with rpc-style WSDL.

#### Service Name

The Service Name will be the value of the name attribute on the service element in the generated WSDL. The exact use of the name depends on products that subsequently use the WSDL such as the SOAP toolkits and UDDI repositories. For example if you subsequently use a SOAP toolkit to generate Java from your WSDL, the Service Name is likely to become the Java interface name.

#### Port Name

This is the name of a specific WSDL port for this service and would usually be derived from the Service Name. One convention would be to provide a Service Name of <xyz> Service and a Port Name of <xyz> Provider.

The Port Name will be the value of the name attribute on the port element in the generated WSDL. The exact use of the name depends on products that subsequently use the WSDL such as SOAP toolkits and UDDI repositories. For example if you use a SOAP toolkit to generate Java from your WSDL, the Port Name could become a Java class name.

#### Port Address

This defines the address at which the service will be made available and must be a valid URL and include the port number if this is different from the default HTTP port. An example of a port address would be:  
`http://localhost:9080/wassoap/servlet/router`



## New WSDL definition wizard: Specify binding properties for SOAP/JMS (optional)

New WSDL definition wizard, specify the binding properties for SOAP/JMS reference material.

### New WSDL definition wizard

List of panels:

- Important information (optional)
- Select a message set folder
- Select destination and format details
- Specify WSDL details
- Select the categories
- Select the bindings
- Specify service details for SOAP/HTTP (optional)
- Specify binding properties for SOAP/JMS (optional)
- Specify service properties for SOAP/JMS (optional)
- Summary of tasks

### Panel properties

#### Destination Style

This field is predefined as 'queue' and cannot be edited.

#### Initial Context Factory

This is the name of a Java class which will allow the SOAP client or server to perform naming and directory service functions through the standard JNDI interface to a particular type of repository.

The following predefined Java classes are offered, or the user can enter their own:

- `com.ibm.websphere.naming.WsnInitialContextFactory` - This corresponds to a repository type of WebSphere Application Server Common Object Services Name Server (part of the CORBA standard)
- `com.sun.jndi.ldap.LdapCtxFactory` - This has a repository type of LDAP (Lightweight Directory Access Protocol)
- `com.sun.jndi.fscontext.RefFSContextFactory` - This has a file system repository type.

The named class must be available on the classpath for a SOAP client or server using this WSDL binding.

If one of these is selected, the corresponding JNDI Provider Type is selected automatically. If a user-defined value is supplied for the Initial Context Factory, the Provider Type field defaults to LDAP.

**Note:** The value of this field will determine the Provider Type and additional details that you will need to provide in the JNDI Provider Type Properties section of this wizard panel.

#### JNDI Connection Factory

This is the JNDI name used to bind to the JMS connection factory and must match your JMS configuration.

**JNDI Destination Name**

This is the JNDI name for the JMS destination factory and must match your JMS configuration.

**JMS Vendor URI**

The JMS vendor URI is a string which uniquely identifies the JMS implementation. This field can not be edited.

**Provider Type**

The Provider Type defines the protocol by which JMS obtains the information it needs.

The Initial Context Factory and the Provider Type dictate what other information you will need to provide in the JNDI Provider Type Properties section of the wizard. This object is set when the Initial Context Factory object is set.

It can be set to:

- ldap
- iiop
- file

If it is set to ldap you will also need to provide:

- Host Name
- LDAP Context Name
- JNDI Provider URL

If it is set to iiop you will also need to provide:

- Host Name
- Port Number
- Target Context
- JNDI Provider URL

If it is set to file you will also need to provide:

- File Name
- Drive Letter
- Path
- JNDI Provider URL

**JNDI Provider Type Properties**

The properties that appear in this section are subject to the setting of the Initial Context Factory and the Provider Type

**Host Name**

The host name or IP address of the machine hosting the JNDI provider

**Port Number**

The port number on the host machine at which the JNDI provider can be contacted

**Target Context**

The JNDI context in which the search is to be performed

**LDAP Context Name**

The JNDI context in which the search is to be performed

**File Path**

The name of the file (including path) that contains the JMS definitions

**Drive Letter**

A drive letter to qualify the JNDI Provider URL. This is only applicable on systems where a drive letter is used to qualify a path.

**JNDI provider URL**

This is the resulting URL as used by JNDI and is read only. It is comprised of the host name and port number and optionally the target context. For example, `iiop://hostname[:port] [/?TargetContext=ctx]` (where [] define what is optional and should not be included in the string).

**New WSDL definition wizard: Specify service properties for SOAP/JMS (optional)**

New WSDL definition wizard, specify the service properties for SOAP/JMS reference material.

**New WSDL definition wizard**

List of panels:

- Important information (optional)
- Select a message set folder
- Select destination and format details
- Specify WSDL details
- Select the categories
- Select the bindings
- Specify service details for SOAP/HTTP (optional)
- Specify binding properties for SOAP/JMS (optional)
- Specify service properties for SOAP/JMS (optional)
- Summary of tasks

**Panel properties****Service Name**

The Service Name will be the value of the name attribute on the service element in the generated WSDL. The exact use of the name depends on products that subsequently use the WSDL such as the SOAP toolkits and UDDI repositories. For example if you subsequently use a SOAP toolkit to generate Java from your WSDL, the Service Name is likely to become the Java interface name.

**Port Name**

This is the name of a specific WSDL port for this service and would usually be derived from the Service Name. One convention would be to provide a Service Name of `<xyz> Service` and a Port Name of `<xyz> Provider`.

The Port Name will be the value of name attribute on the port element in the generated WSDL. The exact use of the name depends on products that subsequently use the WSDL such as the SOAP toolkits and UDDI repositories. For example if you use a SOAP toolkit to generate Java from your WSDL, the Port Name could become a Java class name.

## **New WSDL definition wizard: Summary of tasks**

New WSDL definition wizard: provides a summary of the actions that will occur on finalizing the wizard.

## **New WSDL definition wizard**

List of panels:

- Important information (optional)
- Select a message set folder
- Select destination and format details
- Specify WSDL details
- Select the categories
- Select the bindings
- Specify service details for SOAP/HTTP (optional)
- Specify binding properties for SOAP/JMS (optional)
- Specify service properties for SOAP/JMS (optional)
- Summary of tasks

## **Panel properties**

### **Summary of selections: WSDL files generation**

Provides a summary of what has been requested of the WSDL definition generator.

### **Message definition file selection**

Use the check boxes to select the message definition files from which to generate WSDL definitions.

---

## Part 3. Appendixes



---

## Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.*

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106-0032,  
Japan*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

*IBM United Kingdom Laboratories,  
Mail Point 151,  
Hursley Park,  
Winchester,  
Hampshire,  
England  
SO21 2JN*

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information includes examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not



been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

(C) *(your company name) (year)*. Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *\_enter the year or years\_*. All rights reserved.

---

## Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX	CICS	Cloudscape
DB2	DB2 Connect	DB2 Universal Database
developerWorks	Domino	
Everyplace	FFST	First Failure Support Technology
IBM	IBMLink	IMS
IMS/ESA	iSeries	Language Environment
Lotus	MQSeries	MVS
NetView	OS/400	OS/390
pSeries	RACF	Rational
Redbooks	RETAIN	RS/6000
SupportPac	Tivoli	VisualAge
WebSphere	xSeries	z/OS
zSeries		

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.



---

# Index

## A

- ACORD AL3 messages 682
- attribute group reference
  - CFW properties 204, 234
  - logical properties 178, 234
  - message models, adding to 104
  - TDS format properties 218, 234
  - XML wire format properties 210, 234
- attribute reference
  - CFW properties 204, 235
  - logical properties 178, 234
  - message models, adding to 98
  - TDS format properties 218, 272
  - XML wire format properties 210, 257

## C

- C language
  - importing from C: supported features 722
  - importing message definitions 126
- COBOL
  - importing from COBOL: supported features 724
  - importing message definitions 128
- command line
  - importing message definitions
    - C 127, 130, 135, 138
    - COBOL Copybooks 127, 130, 135, 138
    - WSDL 133
    - XML DTD 127, 130, 135, 138
    - XML Schema 127, 130, 135, 138
- complex type 19
  - content validation properties 182, 279
  - CFW properties 205, 281
  - element, adding to an 102
  - logical properties 179, 275
    - combinations of composition and content validation 280
    - repeats and duplicates 280
  - message models, adding to 100
  - TDS format properties 219, 281
  - XML wire format properties 211, 281
- compound element
  - complex type CFW properties 553
  - complex type logical properties 551
  - complex type TDS format properties 555
  - complex type XML wire format properties 554
  - CFW properties 552, 558
  - logical properties 550, 556
    - value constraints 552, 558
  - TDS format properties 555, 599
  - XML wire format properties 554, 585
- configuration
  - CFW physical properties message model objects 113

- configuration (*continued*)
  - CFW physical properties (*continued*) message sets 85
  - documentation properties message model objects 111 message sets 89
  - logical properties message model objects 110 message sets 83
  - message category file properties 123
  - message model objects 105
  - message set preferences 80
  - physical properties message model objects 112 message sets 84
  - TDS Format physical properties message model objects 115 message sets 87
  - XML Wire Format physical properties message model objects 114 message sets 86
- CFW 41
  - data conversion 43
  - model integrity 42
  - multipart messages 43
  - NULL handling 42
  - NULL handling options 675
  - physical format layers adding 84
  - physical properties
    - configuring for message model objects 113
    - configuring for message sets 85
    - relationship to the logical model 43
- CFW properties
  - attribute group reference 204, 234
  - attribute reference 204, 235
  - complex type 205, 281
  - compound element 552, 558
    - complex types 553
  - deprecated message model objects 552
  - element reference 205, 286
  - embedded simple type 553, 622
  - global attribute 206, 336
  - global attribute group 206, 358
  - global element 206, 361
  - global group 206, 388
  - group reference 206, 392
  - key 207, 394
  - keyref 207, 394
  - local attribute 207, 398
  - local element 208, 456
  - local group 209, 523
  - message 209, 528
  - message model objects 203
  - message sets 150
  - simple type 209, 545
  - unique 209, 546
  - wildcard attribute 210, 547
  - wildcard element 210, 548

## D

- Data element separation 51
- data structures
  - importing 124
- dateTime formats 701
  - component defaults 709
  - CFW binary data 707
  - CFW encoded values 708
  - defaults by logical type 709
  - message set defaults 709
  - string data 702
- deprecated message model objects
  - CFW properties 552
  - logical properties 549
  - physical properties 552
  - properties by object 556
  - TDS format properties 555
  - XML wire format properties 553
- documentation properties
  - message model objects, configuring 111
  - message sets, configuring 89

## E

- EDIFACT messages 679
- element 17
- element reference
  - CFW properties 205, 286
  - logical properties 183, 284
  - message models, adding to 95
  - TDS format properties 222, 328
  - XML wire format properties 211, 313
- element types, changing 108
- elements
  - complex type, adding 102
  - predefined 32
  - self-defining 32
  - simple type, adding 101
- embedded messages 117
- embedded simple type
  - CFW properties 553, 622
  - logical properties 552, 621
  - TDS format properties 555, 651
  - XML wire format properties 554, 649

## F

- file systems
  - importing into workbench 124
- FIX messages 684

## G

- global attribute
  - CFW properties 206, 336
  - logical properties 184, 334
  - message models, adding to 97
  - TDS format properties 223, 341

- global attribute (*continued*)
  - XML wire format properties 212, 336
- global attribute group 25
  - CWF properties 206, 358
  - logical properties 187, 358
- message models, adding to 103
  - TDS format properties 223, 358
  - XML wire format properties 213, 358
- global element
  - CWF properties 206, 361
  - logical properties 187, 359
- message models, adding to 94
  - TDS format properties 223, 366
  - XML wire format properties 213, 361
- global group 23
  - CWF properties 206, 388
  - logical properties 190, 386
- message models, adding to 102
  - TDS format properties 224, 388
  - XML wire format properties 213, 388
- group reference
  - CWF properties 206, 392
  - logical properties 191, 391
- message models, adding to 104
  - TDS format properties 227, 393
  - XML wire format properties 213, 393

## H

- HL7 messages 680
- HTML
  - message set documentation, generating 139

## I

- IBM supplied messages
  - importing message definitions 131
- Import wizard 124
- importing
  - copying and pasting 124
  - dragging and dropping 124
  - from C header files 126
  - from COBOL copybooks 128
  - from command line
    - C 127, 130, 135, 138
    - COBOL copybooks 127, 130, 135, 138
    - WSDL 133
    - XML DTD 127, 130, 135, 138
    - XML Schema 127, 130, 135, 138
  - from IBM supplied messages 131
  - from WSDL 131
  - from XML DTD 134
  - from XML schema 136
  - Import wizard 124
  - message definitions 124

## K

- key
  - CWF properties 207, 394
  - logical properties 192, 394
  - TDS format properties 227, 394
  - XML wire format properties 213, 394

- keyref
  - CWF properties 207, 394
  - logical properties 192, 394
  - TDS format properties 228, 395
  - XML wire format properties 214, 395

## L

- local attribute
  - CWF properties 207, 398
  - logical properties 192, 395
- message models, adding to 97
  - TDS format properties 228, 435
  - XML wire format properties 214, 420
- local element
  - CWF properties 208, 456
  - logical properties 195, 453
- message models, adding to 95
  - TDS format properties 228, 498
  - XML wire format properties 214, 483
- local group
  - CWF properties 209, 523
  - logical properties 198, 521
- message models, adding to 103
  - TDS format properties 229, 524
  - XML wire format properties 215, 524
- logical properties
  - attribute group reference 178, 234
  - attribute reference 178, 234
  - complex type 179, 275
    - combinations of composition and content validation 280
    - content validation 182, 279
    - repeats and duplicates 280
  - compound element 550, 556
  - complex type 551
    - value constraints 552, 558
  - configuring
    - message model objects 110
    - message sets 83
  - deprecated message model objects 549
  - element reference 183, 284
  - embedded simple type 552, 621
  - global attribute 184, 334
  - global attribute group 187, 358
  - global element 187, 359
  - global group 190, 386
  - group reference 191, 391
  - key 192, 394
  - keyref 192, 394
  - local attribute 192, 395
  - local element 195, 453
  - local group 198, 521
  - message 200, 528
  - message model objects 177
  - message set 147
  - simple type 201, 531
    - value constraints 201, 532
  - unique 202, 546
  - wildcard attribute 202, 547
  - wildcard element 202, 548

## M

- message 17
  - CWF properties 209, 528
  - global elements, adding from 94
  - logical properties 200, 528
  - message models, adding to 93
    - TDS format properties 232, 531
    - XML wire format properties 215, 528
- message categories 34
  - member properties 176
  - properties 176
- Message Category editor
  - adding messages to message categories 121
  - message category file properties, configuring 123
  - message category file properties, viewing 123
  - message category files, opening 121
- message category files
  - creating 120
  - deleting 123
- message
  - adding 121
  - deleting 122
  - opening 121
  - properties, configuring 123
  - properties, viewing 123
  - working with 120
- Message Definition editor
  - message definition files, opening 91
  - message model objects, adding 93
  - message model objects, configuring 105
- message definition files 14
  - adding an import 118
  - adding an include 118
  - configuring 105
  - creating 91
  - creating by importing 124
  - deleting 92
  - deleting objects 117
  - imports properties 175
  - includes properties 175
  - linking 118
  - multipart messages 117
  - opening 91
  - properties 174
  - redefines properties 175
  - working with 91
  - XML schema 15
    - extensions 15
    - restrictions 15
- message definitions
  - creating 64
  - generating WSDL
    - relationship to the message model 77
  - importing from C 65
  - importing from COBOL 67
  - importing from IBM supplied messages 75
  - importing from other model representations 64
  - importing from WSDL 72
    - relationship to the message model 73

- message definitions (*continued*)
  - importing from XML DTD 68
  - importing from XML schema 70
    - message sets with namespaces disabled 71
- message model object properties
  - attribute group reference 233
  - attribute reference 234
  - complex type 275
  - compound element 556
  - element reference 284
  - embedded simple type 621
  - global attribute 334
  - global attribute group 357
  - global element 358
  - global group 386
  - group reference 391
  - key 394
  - keyref 394
  - local attribute 395
  - local element 453
  - local group 521
  - message 528
  - simple type 531
  - unique 546
  - wildcard attribute 546
  - wildcard element 547
- message model objects 16
  - adding 93
  - attribute group 25
  - attributes 24
  - complex type 19
  - configuring 105
    - documentation properties 111
    - logical properties 110
    - physical properties 112
  - copying 107
  - CWF properties 203
  - default physical format settings, applying 116
  - deleting 117
  - element 17
  - element type, changing 108
  - group 23
  - identification 27
  - logical properties 177
  - message 17
  - pasting 107
  - physical properties 203
  - properties by object 233
  - renaming 106
  - reordering 106
  - simple types 20
    - list 20
    - restriction 20
    - union 20
  - TDS format properties 218
  - type inheritance 22
  - types 19
  - value constraints 25
  - value constraints, setting 109
  - wildcard attribute 24
  - wildcard element 19
  - working with 92
  - XML wire format properties 210
- message modeling 3
  - advantages of modeling messages 6
  - concepts 4
  - domains, choosing 7
  - formats, choosing 7
  - IDOC domain 9
  - JMS domain 9
  - MIME domain 9
  - MRM domain 8
  - XML domain 8
  - XML messages in the MRM and XML domains 10
- message models 10
  - attribute group reference, adding 104
  - attribute reference, adding 98
  - complex type, adding 100
  - developing 3
  - documentation, generating 78
  - element reference, adding 95
  - global attribute group, adding 103
  - global attribute, adding 97
  - global element, adding 94
  - global groups, adding 102
  - group reference, adding 104
  - local attribute, adding 97
  - local element, adding 95
  - local group, adding 103
  - message categories 34
  - message definition files 14
  - message dictionary, generating 75
  - message set projects 11
  - message sets 11
    - IBM supplied message sets 13
    - identification 12
    - recommendations 12
    - resources 12
    - versions and keywords 13
  - message, adding 93
  - message, adding from global elements 94
  - model integrity 39
  - model representations, generating 75
  - namespaces 35
    - non-XML messages 37
    - reusing message definition files 38
  - object cardinality 32
  - simple type, adding 99
  - substitution groups 33
  - task list errors
    - applying a quick fix 90
    - quick fix list 710
  - wildcard attribute, adding 98
  - wildcard elements, adding 96
  - WSDL, generating 77
  - XML Schema, generating 76
- Message Set editor
  - documentation properties, configuring 89
  - logical properties 83
  - message sets, opening 81
  - physical formats configuring 84
- message set projects
  - creating 79
  - deleting 80
  - working with 79
- message sets 11
  - configuring
    - CWF properties 85
    - documentation properties 89
    - logical properties 83
    - physical format layers 84
    - preferences 80
    - TDS Format properties 87
    - XML Wire Format properties 86
  - creating 81
  - CWF layers
    - adding 84
  - CWF properties 150
  - default physical format settings, applying 88
  - deleting 90
  - documentation properties 172
  - generating information from 139
  - IBM supplied message sets 13
  - identification 12
  - importing
    - from C: supported features 722
    - from COBOL: supported features 724
    - from WSDL: generated objects 729
    - from WSDL: restrictions 729
    - from XML DTD: unsupported features 734
    - from XML schema: unsupported features 732
    - supported and unsupported features 722
  - keywords, guidance for defining 172
  - logical properties 147
  - opening 81
  - physical format layers
    - adding 84
    - removing 89
    - renaming 88
  - preferences 145
  - editors 145
  - validation 146
  - XML schema importer 147
  - recommendations 12
  - resources 12
  - TDS Format layers
    - adding 86
  - TDS format properties 162
    - defaults 169
    - TDS mnemonics 168, 690
  - versions and keywords 13
  - working with 80
  - XML Wire Format layers
    - adding 85
  - XML wire format properties 155
    - In-line DTDs and the DOCTYPE text property 162
- messages
  - embedding 117
  - message category file
    - adding to 121
    - deleting from 122
  - multipart 28
  - identifying using Message Identity 28

- messages (*continued*)
  - multipart (*continued*)
    - identifying using Message Path 31
    - predefined 32
    - self-defining 32
- MIME domain 712
  - parser restrictions 715
  - parser use 715
  - standard header fields 712
- MRM domain 673
  - additional CWF information 675
    - data conversion 675
    - NULL handling options 675
  - additional logical format
    - MRM model restrictions 674
  - additional logical information 673
  - additional TDS format
    - information 679
      - industry standard formats 679
      - message characteristics in the MRM 684
      - message model integrity 692
      - NULL handling options 691
      - regular expressions to parse data elements 695
    - additional XML wire format
      - information 676
        - NULL handling options 676
  - MRM: Generated model
    - representations 716
      - document generation 716
      - WSDL generation 716
      - XML schema generation 719
  - multipart messages
    - creating 117

## N

- namespaces 35
  - non-XML messages 37
  - reusing message definition files 38
- namespaces in the MRM domain 35
- New Message Category File wizard 120
- New Message Definition File wizard 91
- New Message Set Documentation wizard 139
- New Message Set Project wizard 79
- New Message Set Web Service Definition wizard 141
- New Message Set wizard 81
- New XML Schema wizard 140
- NULL handling
  - CWF options 675
  - TDS format options 691
  - XML wire format options 676
    - NULL element and NULLValAttr 677
    - NULL representation for Binary data 678
    - NULL value 677

## P

- performance
  - regular expressions to parse TDS messages 701
- physical format layers 39
  - CWF 41
    - data conversion 43
    - model integrity 42
    - multipart messages 43
    - NULL handling 42
    - relationship to the logical model 43
  - CWF layers
    - adding 84
  - default settings, applying 88
  - message model object properties, configuring 112
  - message sets
    - adding to 84
    - removing 89
    - renaming 88
  - TDS Format 49
    - data conversion 62
    - data element separation 51
    - model integrity 60
    - multipart messages 61
    - NULL handling 61
    - relationship to the logical model 62
  - TDS Format layers
    - adding 86
  - XML Wire Format 45
    - data conversion 47
    - model integrity 46
    - multipart messages 47
    - NULL handling 46
    - relationship to the logical model 47
    - xsi:type attributes 48
- XML Wire Format layers
  - adding 85
- physical formats, applying default settings to message model objects 116
- physical properties
  - configuring
    - message model objects 112
    - message sets 84
  - deprecated message model objects 552
  - message model objects 203
- preferences
  - message sets 145
    - configuring 80
    - editors 145
    - validation 146
    - XML schema importer 147
- properties
  - deprecated message model objects 549
  - documentation
    - message sets 172
  - message categories 176
  - message category members 176
  - message definition file imports 175
  - message definition file includes 175
  - message definition file redefines 175
  - message definition files 174

- properties (*continued*)
  - message model objects 177
  - message sets, documentation 172

## S

- simple type 20
  - CWF properties 209, 545
  - element, adding to an 101
  - list 20
  - logical properties 201, 531
    - value constraints 201, 532
  - message models, adding to 99
    - restriction 20
  - TDS format properties 232, 545
  - union 20
  - XML wire format properties 218, 545
- SWIFT messages 681

## T

- TDS format 49
  - data conversion 62
  - data element separation 51
    - data pattern separation types 59
    - delimited separation types 55
    - fixed length separation types 52
    - tagged separation types 53
  - message model integrity 692
    - general rules 692
    - omission and truncation of elements 694
    - restrictions for nesting complex types 694
  - model integrity 60
  - multipart messages 61
  - NULL handling 61
  - NULL handling options 691
  - physical format layers
    - adding 86
  - physical properties
    - configuring for message model objects 115
    - configuring for message sets 87
  - regular expressions to parse data elements 695
    - multiple delimiters 699
    - performance considerations 701
    - syntax 697
    - variable number of repeats 700
  - relationship to the logical model 62
- TDS format properties
  - attribute group reference 218, 234
  - attribute reference 218, 272
  - complex type 219, 281
  - compound element 555, 599
    - complex types 555
  - deprecated message model objects 555
  - element reference 222, 328
  - embedded simple type 555, 651
  - global attribute 223, 341
  - global attribute group 223, 358
  - global element 223, 366
  - global group 224, 388
  - group reference 227, 393

TDS format properties (*continued*)  
key 227, 394  
keyref 228, 395  
local attribute 228, 435  
local element 228, 498  
local group 229, 524  
message 232, 531  
message model objects 218  
message set defaults 169  
message sets 162

    TDS mnemonics 168, 690  
simple type 232, 545  
unique 233, 546  
wildcard attribute 233, 547  
wildcard element 233, 549  
TDS industry standard formats 679  
    ACORD AL3 messages 682  
        fixed length AL3 683  
        tagged encoded length to support  
        reversioning 684  
    EDIFACT messages 679  
    FIX messages 684  
    HL7 messages 680  
    SWIFT messages 681  
    TLOG messages 682  
    X12 messages 682  
TDS message characteristics in the  
MRM 684  
    data element separation 686  
    special characters to model a  
    message 688  
        mnemonics as special  
        characters 689  
        TDS mnemonics 168, 690  
TLOG messages 682  
trademarks 777

## U

unique  
    CWF properties 209, 546  
    logical properties 202, 546  
    TDS format properties 233, 546  
    XML wire format properties 218, 546

## V

value constraints 25  
    setting 109

## W

Web Service Definitions  
    message set, generating from 141  
wildcard attribute 24  
    CWF properties 210, 547  
    logical properties 202, 547  
    message models, adding to 98  
    TDS format properties 233, 547  
    XML wire format properties 218, 547  
wildcard element 19  
    CWF properties 210, 548  
    logical properties 202, 548  
    message models, adding to 96  
    TDS format properties 233, 549  
    XML wire format properties 218, 548

WSDL 141  
    importing from WSDL  
        generated objects 729  
        restrictions 729  
    importing message definitions 131  
    relationship to the message model  
        generating WSDL 77  
    importing WSDL 73

## X

X12 messages 682  
XML DTD  
    importing message definitions 134  
XML namespaces in the MRM  
    domain 35  
XML rendering options 678  
XML schema  
    importing 136  
    message set, generating from 140  
XML Schema  
    message editor only features 674  
XML wire format  
    NULL handling options 676  
        NULL element and  
        NULLValAttr 677  
        NULL representation for Binary  
        data 678  
        NULL value 677  
    XML rendering options 678  
XML Wire Format 45  
    data conversion 47  
    model integrity 46  
    multipart messages 47  
    NULL handling 46  
    physical format layers  
        adding 85  
    physical properties  
        configuring for message model  
        objects 114  
        configuring for message sets 86  
    relationship to the logical model 47  
    xsi:type attributes 48  
XML wire format properties  
    attribute group reference 210, 234  
    attribute reference 210, 257  
    complex type 211, 281  
    compound element 554, 585  
        complex types 554  
    deprecated message model  
    objects 553  
    element reference 211, 313  
    embedded simple type 554, 649  
    global attribute 212, 336  
    global attribute group 213, 358  
    global element 213, 361  
    global group 213, 388  
    group reference 213, 393  
    key 213, 394  
    keyref 214, 395  
    local attribute 214, 420  
    local element 214, 483  
    local group 215, 524  
    message 215, 528  
    message model objects 210  
    message sets 155

XML wire format properties (*continued*)  
    In-line DTDs and the DOCTYPE  
        text property 162  
    simple type 218, 545  
    unique 218, 546  
    wildcard attribute 218, 547  
    wildcard element 218, 548









Printed in USA