WebSphere Message Broker

Message Models

Version 6 Release 1

WebSphere Message Broker

Message Models

Version 6 Release 1

Note
Before you use this information and the product that it supports, read the information in the Notices appendix.

This edition applies to version 6, release 1, modification 0, fix pack 6 of IBM WebSphere Message Broker and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this topic collection

This PDF file has been created from the WebSphere Message Broker Version 6.1 (fix pack 6 update, March 2010) 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 result in a "file not found "error message. 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

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 when you might want to model messages, and describe the message modeling objects that you can use, such as message sets and message definition files.

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

The following tasks are provided for developing message models:

- "Working with a message set project" on page 81
- "Working with a message set" on page 82
- "Working with a message definition file" on page 94
- "Working with message model objects" on page 97
- "Creating a multipart message" on page 123
- "Linking from one message definition file to another" on page 124
- "Working with a message category file" on page 125
- "Working with data structures" on page 129
- "Generating documentation from message sets and message flows" on page 143
- "Generating XML Schemas" on page 143
-
- "Generating a WSDL definition from a message set" on page 146

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

The workbench lets you open resource files with other editors. However, use only the workbench to edit resource files that are 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 that have a defined structure that is known and agreed by the sender and the receiver.

Applications typically use a combination of messages, including those that are defined by the following structures or standards:

- 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 so that they can be understood by WebSphere Message Broker message flows.

When 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.

The following topics together give an overview of Message modeling:

- "Message modeling concepts"
- "Why model messages?" on page 6
- "Message domains and parsers" on page 7
- "The message model" on page 7
- "Physical formats in the MRM domain" on page 39
- "Ways to create message definitions" on page 66
- "Generate model representations" on page 78

You can import either of the following samples to explore message set projects to understand how the sample's messages are modeled in different formats.

- Video Rental
- Comma Separated Value (CSV)

The following samples from the Samples Gallery also have message sets supplied:

- EDIFACT
- FIX
- SWIFT
- X12

You can view samples only when you use the information center that is integrated with the Message Broker Toolkit.

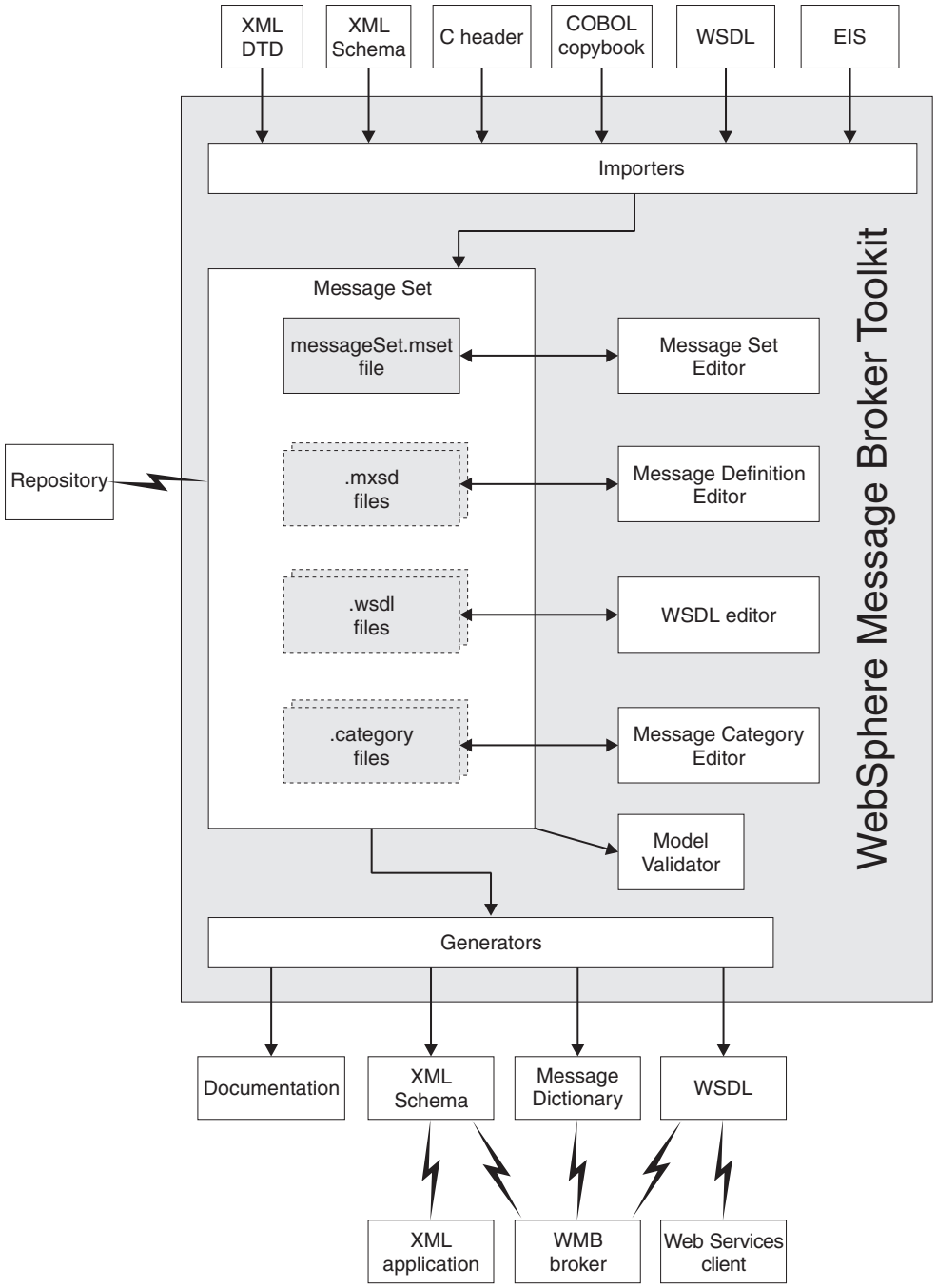
Message modeling concepts

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

When you have created your message models, include them in your broker archive (BAR) file with the message flows that use those models. Deploy the BAR file to the broker, which uses your message models to automatically parse and write your message formats.

When you model messages, you must understand the following concepts:

- *Message set projects*
- *Message sets*
- *Message definition files*
- *Web Services Description Language (WSDL) 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 that are 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). A message set contains the following files:

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

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

When you have created a message set, you typically import application message formats described by XML DTD, XML Schema, WSDL files, C structures, COBOL structures, or EIS systems, 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 a broker, parser, or application. This might be in one of the following forms:

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

Messages can be optionally grouped into *message categories* for convenience . 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 that you are creating follows certain rules. There are rules for both the logical structure and the physical formats. This 'model validation' ensures the integrity of your model, but does not necessarily prevent you from saving a message model file that is not valid.

WebSphere Message Broker supplies a range of parsers to parse and write message formats. Each parser is suited to a particular class of messages (for example, fixed-length binary, delimited text, or XML) known as a *message domain*. When you create a message set, you specify which domains the message set supports. This determines which parsers can be used when you parse and write messages that are defined within that message set.

Why model messages?

WebSphere Message Broker 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. Most message formats, however, are not self-defining, and the parser must have access to a predefined model that describes the message, if it is to parse it correctly.

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.

Examples of messages that do not have a self-defining format are binary messages that originate from a COBOL program, and from SWIFT formatted text messages. Neither contain sufficient metadata to enable a parser to understand the messages.

Even if your messages are self-defining and do not require modeling, the following advantages of modeling them might be useful:

- Runtime validation of messages. Without a model, a parser cannot check whether input and output messages have the correct structure and data values.
- 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, 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 operations on 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 cannot use the Mapping editor.
- Reuse of message models, in whole or in part, by creating additional messages that are based on existing messages.
- 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 that are offered by WebSphere Message Broker, model your message formats.

To speed up the creation of message models, importers are provided to read metadata such as C header files, COBOL copybooks, XML Schema and DTDs, WSDL files, and EIS metadata, and create message models from that metadata. Additionally, predefined models are available for common industry standard message formats such as SWIFT, EDIFACT, X12, FIX, HL7, and TLOG.

Message domains and parsers

WebSphere Message Broker supplies a range of *parsers* to parse and write message formats.

A parser is invoked when the bit stream that represents an input message is converted to the internal form that can be handled by the broker. The internal form, a logical tree structure, is described in Logical tree structure. Similarly, a parser is invoked to convert a logical tree back into a bit stream.

Each parser is suited to a particular class of messages (for example, fixed-length binary, delimited text, or XML) known as a *message domain*.

When you create a message set, you specify which message domains the message set supports. This determines which parsers are used when you parse and write messages that are defined within that message set.

The parsers that are supplied with WebSphere Message Broker are described in Parsers.

The message model

The message model consists of the following components.

- Message set projects
- Message sets

- Message definition files
- WSDL 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 that defines a language for describing the structure of XML documents. It is ideally suited to describing the messages that 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 structure of all kinds of message format, not just XML.

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

To understand the different ways that you create and populate message definition files, see “Ways to create message definitions” on page 66. See “Physical formats in the MRM domain” on page 39 for a description of the physical formats that are 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. If the message set is modeling messages from EIS systems, the name of the message set project provides the name of the message set and, optionally, a single Adapters folder. You can create a message set project using the following methods:

- The New Message Set wizard.
- The Quick Start wizards.

These restrictions apply to message set projects:

- A message set project must contain just one message set.
- A message set project cannot refer to any other message set.

Import either of the following samples from the Samples Gallery to see how message set resources are stored in a message set project. The sample's message flow resources are stored separately in a Message Flow project.

- Video Rental
- Comma Separated Value (CSV)

You can view samples only when you use the information center that is integrated with the Message Broker Toolkit.

Message sets overview

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.

When you create a new message set, a new message set project is automatically created with a name that is the same as that of the message set.

You can base your new message set on an existing message set. In this case, 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 the following key properties:

Supported message domains
The message domains that are supported by the message set. The supported domains determine what is generated for deployment to a broker, and are used when parsing and writing the messages that are defined within the message set.

Default message domain
The default domain of the message set.

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

Message set resources:

Resources in a message set are created as files, and are displayed under the message set folder in the Broker Development view.

- Message set file `messageSet.mset`

  There is always one, and only one, `messageSet.mset` file in a message set. This file 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), Tagged Delimited String Format (TDS), and XML Wire Format (XML).

  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 the associated physical model, in XML Schema form, for a group of related messages.
- WSDL files that have the suffix `.wsdl`

  These files are used by the SOAP domain. You can have many WSDL files in a message set.
- 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 content of the message set in a form that can be used by a broker parser or an application. This might be:

- a message dictionary for deployment to a broker

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

Message set identification:

A message set is identified by the name of the message set folder in the message set.

When you need to refer to a message set from a message flow (for example, when setting the Message Set property of an input node), use the message set name.

A message set also has a 13-character identifier that is guaranteed to be unique. You can use this identifier, instead of the message set name, to refer to a message set, but only if you are using the MRM or IDOC domains. Other domains do not recognize the identifier.

A message set also has an alias. An alias can only be used with MRM multipart messages.

Message set recommendations:

You can have as many message definition files as you want within one message set, but you should 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.
- MRM physical formats apply to *all* objects within the message set.

  Therefore, for example, if you are using the MRM domain and have an XML message and an unrelated CWF message modeled in the same message set, CWF physical format properties are present for all objects. But the CWF properties are of no interest to the XML message and therefore take default values in those objects. This can result in unwanted task list warnings.
- Recursion is not permitted for MRM CWF and TDS physical formats.

  Therefore, 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 MRM CWF or TDS physical formats.

Message set version and keywords:

When you develop a message set, you can define the version of the message set, and other key information that you want to be associated with it.

After you have deployed the message set in a BAR file, you can view the message set properties in the workbench. The properties include the deployment and modification dates and times (the default information that is displayed), and the additional version or keyword information that you have set.

You can define information to give details of 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.

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

Keywords

You must define keywords in the Documentation property of the message set. Keywords follow certain rules to ensure that the information can be parsed. The following example shows the type of information that you can define in the Documentation property:

```
$MQSI Author=John Smith MQSI$
```

The following table contains the information that is displayed by the workbench:

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

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

Restrictions within keywords

Do not use the following characters within keywords, because they cause unpredictable behavior:

```
^ $ . | \ < > ? + * = & [ ] ( )
```

You can use these characters in the values that are associated with keywords; for example:

- `$MQSI RCSVER=$id$ MQSI$` is acceptable
- `$MQSI $name=Fred MQSI$` is not acceptable

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* that 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 "Importing from other model representations to create message definitions" on page 67.

A message definition file can be associated with a namespace, so that all message model objects that are 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 "Namespaces in the message model" on page 33.

One message definition file can reuse message model objects that are 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 37.

XML Schema and the message model:

XML Schema is an international standard that defines a language for describing the structure of XML documents.

The XML Schema language is ideally suited to describing the messages that flow between business applications, and it is widely used in the business community for this purpose.

WebSphere Message Broker uses XML Schema 1.0 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 messages.

Some important restrictions and extensions of XML Schema exist. These are documented in "Schema restrictions in the message model" and "Schema extensions in the message model."

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 in the message model:

Some XML Schema 1.0 features are not supported in the message model.

Unsupported XML Schema features: The following feature is accepted, but not supported, and causes validation errors if it is used in your message model:

- Redefines

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 in the message model:

The message model provides some facilities that are not specified in the XML Schema 1.0 specification.

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

Composition: The message model adds the following compositions that are beyond the XML Schema 1.0 specification:

message
A refinement of *choice* that is allowed to contain only 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.

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

unorderedSet
A set of elements that can appear in any order. Groups are not allowed within an *unorderedSet*. Unlike an *all* group, elements within an *unorderedSet* are allowed to repeat. However, duplicate elements are not allowed.

Compositions *orderedSet* and *unorderedSet* allow message models that were produced in earlier versions of the product to be supported.

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 parsed and 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 objects that make up a message model.

Message
A *message* describes the structure and content of a set of data that is exchanged between applications that send and receive the data. A *message* is a special kind of complex element.

Simple element
A *simple element* describes one or more named data fields in a message. It is based on a *simple type* (for example, string, integer or float). A simple element 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* describes a named complex structure within the message. The content of a complex element is defined by a *complex type*. A complex element can repeat.

Complex type
A *complex type* describes a complex structure within a message. It contains *elements* (simple or complex), *attributes*, and *groups* that are organized into a tree-like hierarchy.

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

Attribute
An attribute describes an XML attribute. Attributes are very similar to simple elements, but they require special treatment when used with XML messages. In messages that are not XML messages, attributes are typically not used, but if they do appear they are treated exactly like a simple element based on the same simple type.

Global and local objects: Most 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.

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 makes 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 definition 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 an object 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 that you might require. Text entered here does not affect the processing of messages in any way.

Message model objects: messages:

A *message* describes the structure and content of a set of data that is passed from one application to another.

A message consists of elements that are 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 message data can be parsed into a logical tree and manipulated easily by the broker.

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

Multipart messages: If necessary, a message can contain other messages. This is necessary for modeling certain large and complex messaging standards such as SWIFT and EDIFACT. Such a message is known as a *multipart message*. The contained messages are known as *embedded messages*.

Message identification: Messages are identified by their name or an alias. The alias is an optional user-specified string that identifies the (multipart) message. The name and 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 global element declaration. Extra information is provided by XML Schema annotations on the element declaration.

Message model objects: elements:

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

An element has a specific meaning that is 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 or Java in a message flow node.
- Is further defined by its type; for example, the type defines the range of values that an element can have.
- Can be defined globally or locally.

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* that defines its content.

A complex element is a named structure that contains 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* that 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 30.

Default and fixed values: An element can be given a default value, so that if no value is supplied by the message, the default value is used. Alternatively, a fixed value can be defined, and the element always takes that value. The precise use of default and fixed values is dependent on the message domain.

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 "Message model objects: simple types" on page 17. The XML Schema term for a value constraint is a *facet*.

Defining substitution groups: If you are modeling XML messages, 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 in the message model" on page 31.

Message model objects: 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 that contain a hierarchy of other elements.

For more information, see:

- "Message model objects: simple types" on page 17
- "Message model objects: complex types"
- "Message model objects: type inheritance" on page 19

Message model objects: 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 that contains 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 "Message model objects: groups" on page 20.

Wildcards
Complex types can contain wildcard elements, which allow unmodeled elements to appear within any elements that are 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 that are based on the complex type.

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

Composition: The composition of a complex type describes how the members of the type are organized. For more information, see "Message model objects: groups" on page 20.

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 "Message model objects: groups" on page 20.

Substitution settings: A complex type has parameters that 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 in the message model" on page 31 and "Message model objects: type inheritance" on page 19.

Message model objects: 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 that are based on simple types) cannot contain attributes or child elements. Simple types stand in contrast to complex types, which define the structure of an element, but typically 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 have a name.

Variations of simple types

Built-in

XML Schema defines a large number of simple types for you to use, covering all the standard data types such as strings, integers, decimals, and floats.

Restriction

You can define your own simple types by deriving from another simple type (the base type) by restriction. A restriction type can have value constraints applied to it.

A restriction type can derive from a built-in simple type or a user-defined simple type.

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 (see below). 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 results 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 to 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 that uses them, but issues a warning that enumeration and pattern facets are ignored by the broker.

Union

A union type is 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 comprise a union type are known as its member types. There is no upper limit on how many member types can exist, but there must be at least one. A member type can 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.

MRM domain

The MRM parser does not apply value constraints until the data is in the logical tree. This means that it is not possible to choose between two simple types that are derived from the same fundamental type, but have different value constraints (for example, an integer in the range 1-10 and an integer in the range 11-20). A warning appears in the task list if this is attempted, and the parser ignores the value

constraints when it resolves 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 that uses them, but the editor issues a warning that enumeration and pattern facets are ignored by the MRM parser.

Value constraints

Any value constraints that are applied to the derived type must further restrict the base type (and any elements based on it). It is not valid for a derived type to weaken or remove a value constraint that its base type has defined. If no value constraints are applied to the derived type, the derived type is almost identical to its base type, but it is treated as a restriction of the base type in situations where that is relevant (type inheritance and element substitution).

Message model objects: 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 about 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 valid values (or valid 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 valid values (or valid type members) than elements of the base type.

- For example, an extension of a complex type might add type members that 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 that is based on the simple type; you cannot extend a simple type by widening its range of valid values.

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

However, a complex type can extend a simple type. This does not affect the range of valid 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 that contains 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 not valid to derive another complex type from this type by restriction.
- extension: It is not valid to derive another complex type from this type by extension.
- all: It is not valid 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 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.

Message model objects: groups:

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

Groups can be ordered (sequence or orderedSet) unordered (all or unorderedSet), or selective (choice or message). 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 that 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 substructure 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 that 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 modelled. Wildcard elements can be included to allow unmodelled 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 that 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 have a name, and are displayed using the group's composition by the message definition editor.

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

- A sequence is a set of elements that must appear in the same order as they are listed.
- An all group is a set of elements that 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: orderedSet, unorderedSet, and message. For more information, see "Schema extensions in the message model" on page 12.

Content validation: The Content validation property is applied only if the domain is MRM or IDOC, and if validation is enabled.

Content validation determines how strictly the content of the group should be validated. See "MRM content validation" on page 194 for more details.

Allowable values of the Content validation property are:

Closed
: The contents of the group are validated strictly against the model. Only elements that are defined as children of the group can appear as children.

Open Defined
: Elements that are declared within the same message set can appear as children of the group, even if they are not defined as children.

Open Any elements can appear as children of the group.

The Content validation property does not affect validation in the XMLNSC or SOAP domains. Validation in these domains follows the rules of XML Schema 1.0.

Message model objects: attributes:

An *attribute* describes an XML attribute.

Attributes are provided to simplify the modeling of XML messages; if none of your messages use the XML physical format, use 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 that 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 are treated in exactly the same way as 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 cannot be used elsewhere in the message model.

Optional attributes: 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 30.

Default and fixed values: An attribute can be given a default value so that, if the attribute is missing from the message, the default is used. Alternatively, a fixed value can be defined, and the attribute always takes that value. The precise use of default and fixed values is domain dependent.

Value constraints: An attribute's value can be constrained by using *value constraints*, which define the range of legal values for the attribute. Value constraints are associated with the simple type on which the attribute is based. For more details, see "Message model objects: simple types" on page 17. In XML Schema, the term for *value constraint* is *facet*.

Message model objects: wildcard elements:

A wildcard element represents an element that does not appear in the message model, but which could appear at the same position as the wildcard element in the 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.

The Process Content and Namespace properties control the namespace to which elements appearing in place of the wildcard element must belong.

MRM domain

If you enable validation in your message flow, and your message is in the MRM domain, wildcard elements are validated against the model according to the following rules:

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

Message model objects: wildcard attributes:

A *wildcard attribute* allows unmodelled attributes to appear in a message.

The Process Content and Namespace properties control the namespace to which attributes that appear in place of the wildcard must belong.

MRM domain

If you enable validation in your message flow, and your message is in the MRM domain, wildcard attributes are 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 behavior of the XMLNSC domain in validating mode.

Message model objects: attribute groups:

An *attribute group* defines a set of attributes that 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 that use them.

Message model objects: simple type value constraints:

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

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

The value constraints that are discussed here are modeled by XML Schema facets, and are associated with a simple type.

Tip: If the message set is deployed to WebSphere Message Broker, elements and attributes can be validated against value constraints, so that violations are reported as errors or warnings. The XMLNSC domain uses all the different types of value constraint when validating. The MRM domain uses a subset; the restrictions are noted below.

Types of value constraint:

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 that 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 that are defined for a simple type must not relax the constraints that are imposed by any of its ancestor types. For example, a type 'longString' (Max Length=100) cannot be derived from a type 'shortString' (Max Length=10).

Note: For the MRM domain, 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 Broker treats Length facet as MaxLength.

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

Range constraints specify the allowable range of values for all elements that 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 that are numerical, or that relate to calendar and time values. They cannot be applied to strings, because the ordering of string values depends on the character set that is used.

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

Note: For the MRM domain, exclusive constraints cannot 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 that 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 that are defined for a simple type must not increase the range of allowed values. For example, a type 'AllColors' (with enumerations for all colors of the rainbow) cannot be derived from a type 'MonoColors' (with enumerations for 'black' and 'white' only).

Precision constraints : Total Digits and Fraction Digits

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

Precision constraints are inherited from ancestor types, and any precision constraints that are defined for a simple type must not relax the constraints

that are imposed by any of its ancestor types. For example, a type 'veryPrecise' (Fraction Digits=10) cannot be derived from a type 'notVeryPrecise' (Fraction Digits=1).

Note: For the MRM domain, the broker applies these constraints only to xsd:decimal and user types that are derived from it; precision constraints that are applied to an integer simple type are ignored.

Pattern constraints

A pattern constraint is a regular expression that specifies a set of allowed values for all elements that are 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 that are based on the simple type; that is, all the patterns are combined by 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 that are contributed by each ancestor type must be satisfied, as well as the set that is contributed by the simple type itself; that is, the sets of pattern constraints from each level in the type hierarchy are combined by using Boolean AND.

Note: For the MRM domain, pattern constraints can be applied only to simple types that are derived from xsd:string.

White space constraints

A white space constraint specifies how a parser should treat white space for all elements that are based on the simple type.

Note: For the MRM domain, white space constraints are not applied. Although the MRM physical formats allow white space to be precisely controlled for each physical format that is defined for the message, these physical properties are separate from the white space constraint in the logical model, and are not used for validation purposes.

Message 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 the message set can specify a namespace. Namespaces are an XML Schema mechanism for organizing groups of related objects into a named 'module'.

Global objects in different namespaces are allowed to share the same name. Therefore, 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.

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 that are allowed to appear at that location in the message structure, or it might be empty, allowing any message to appear. When a message is parsed, only one message can appear in that position as an embedded message.

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 EDIFACT, X12, SWIFT, SOAP XML, SAP ALE IDoc, multipart MIME, and RosettaNet.

Identifying the embedded message: When a multipart message is parsed, the parser must be able to identify the embedded message; it might be any of the messages that are referenced by the group or complex type, or it might be a message that is not referenced by the group or complex type, perhaps from a different message set. This is achieved using one of four techniques, *Automatic*, *Message Identity*, *Message Path*, or *Manual*.

Automatic

Used when parsing XML messages, such as SOAP. The parser automatically identifies and parses embedded messages using the tag in the XML document.

Message Identity

Used by the MRM parser. See “Identifying an embedded message using a Message Identity” on page 27.

Message Path

Used by the MRM parser. See “Identifying an embedded message using a Message Path” on page 29.

Manual

Used by the MIME parser. The parser treats embedded messages as BLOBs. If you want to parse the BLOB using another parser, you must do so manually using ESQL, or Java, or a ResetContentDescriptor node.

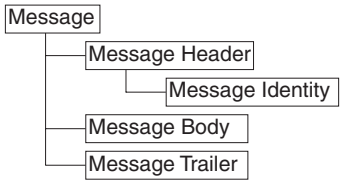
Restrictions: Unless using the *Manual* identification technique, all embedded messages must be of the same physical format as the outermost message, and have the same character set and encoding.

When using the *Automatic* or *Message Path* identification techniques, all embedded messages must be from the same message set as the multipart message.

Identifying an embedded message using a Message Identity:

You can identify an embedded message by using the Message Identity. This technique is used by the MRM domain, and 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 the diagram:



In this example, the Message Header and Message Trailer act as an envelope for the message body. They typically 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, 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 that corresponds to either the Name or the Message Alias of the next embedded message in the bit stream. 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, use the Message Alias property to specify this value. The MRM parser tries to match on Name first, and if that fails, it tries 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, Message Identity is used. You should only use 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 that is defined within 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 that corresponds to either the Identifier, Name, or Message Set Alias of the message set that defines the next embedded message in the bit stream. This enables the MRM parser to correctly identify the message set to use.

If the Message Set Identity element value does not match the Identifier or Name of the message set, use the Message Set Alias property to specify this value. The MRM parser tries to match on Identifier first, then on Name, and finally on 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 that contains `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

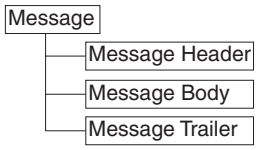
Both Message Identity and Message Set Identity are applicable to all physical formats. Versions of the TDS physical format prior to Version 6.0 included embedded message identification by Message Key, which worked in a similar manner to Message Identity. Message Key has been deprecated and is superseded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

This technique is used by the MRM domain.

In the diagram, 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.



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, 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 be:

```
Message/Message Body
```

If the path to the innermost message contains intermediate elements, these intermediate elements must also be included in the path. 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 to produce a final path that is used to locate embedded messages.

Message Identity takes priority over Message Path. If both are specified, Message Identity is used. Use only one of these techniques for a given multipart message.

You cannot use the Message Path technique to identify multiple peer embedded messages.

Embedded messages defined in different message sets

This option is not supported by the Message Path technique.

Physical format considerations

The Message Path technique is applicable to all physical formats.

Cardinality: optional, repeating and mandatory 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.

Elements: A *mandatory* element has Min Occurs>= 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 Max Occurs=-1, which indicates that an unlimited number of repeats are allowed. A repeating element 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 that refer to the same element, the second reference is a *duplicate*. This is different from a repeating element, because the two references are typically separated by other members of the type or group. In the input message, the second occurrence typically does not appear immediately after the first occurrence. Duplicate element references are not allowed within types and groups that 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 for the parser 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.

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 untagged delimited (for example, comma separated), you must ensure that your message model defines every message and every element that 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 are self-defining. In this situation, you cannot use message definitions to influence the parsing and writing of elements; the self-defining elements are parsed and written according to the default behavior 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, and all data is assumed to be string type.

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 that have been defined. This provides a range of benefits, all of which arise from the information that is provided to the broker through the message model.

Substitution groups in the message model:

Substitution groups are an XML Schema feature that provides a way of substituting one element for another in an XML message.

A substitution group is a list of global elements that 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.

Tip: If your messages are never rendered as XML, or if you have a simple message model, use 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 that 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. Typically, abstract elements 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 that 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 that is based on a restriction of the element's type cannot substitute for the element.
- If the block attribute contains extension, an element that is based on an extension of the element's type cannot substitute for the element.
- If the block attribute contains substitution, an element that 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 that 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 that 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 that 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 that 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 that 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 is 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 “Generate WSDL” on page 80.

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 in the message model

Namespaces provide a method to qualify object names.

XML instance documents and XML Schemas can use namespaces.

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 that 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 that are 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. Therefore, namespaces assist in the development of a library of XML Schemas that can be developed independently. If the namespace name that is used for an XML Schema is unique, you do not have to be concerned about name clashes with objects that are 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 must be unique. You might be more familiar with the concept of a Universal Resource Locator (URL). URIs often use the same syntax as URLs, but 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 that belong to that namespace. The prefix to be substituted for a namespace in an XML instance document or XML Schema is specified using an xmlns attribute. A default namespace can also be defined using an xmlns 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 not qualified by a namespace.

The message model
The message model provides the ability to support namespaces within message sets. However, you can choose whether to enable or disable namespaces for your message set. If you disable namespaces when you create your message set, you can enable namespaces later. However, when 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 has an associated namespace, or whether it is in the notarget namespace. If you 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 that are defined within a Message Definition File can reference objects in other Message Definition Files within the same message set. To do this, import or include one Message Definition file within another Message Definition File.

Message parsing and message flows

WebSphere Message Broker parsers that are namespace aware recognize prefixed names in the XML messages that they parse, and internally map these to the correct namespace. Elements and attributes can be either qualified or not qualified with a namespace, as discussed in the message model section.

If you are using XML format in the MRM domain, elements or attributes are matched, based on the namespace in the dictionary when the parsed message is matched against the dictionary that is generated from the message model. Therefore, for an element or attribute in a message to match with the dictionary, both its name and its namespace must match.

If you are using the DataObject domain, the SOAP domain, or the XMLNSC domain (in validating mode), elements or attributes are matched, based on the namespace in the XML Schema when the parsed message is matched against the XML Schema that is generated from the message model. Therefore, for an element or attribute in a message to match the XML Schema, both its name and its namespace must match.

Support is provided that allows you to specify namespaces when writing ESQL or Java. It is not necessary to write ESQL or Java that is namespace aware if you are not using namespaces. However, if you decide to use namespaces, your message definition files can target any namespace that you choose, and it is necessary to write namespace-aware ESQL or Java. The namespace in which an element resides is stored in the message tree when parsed. This is a logical property and it is held regardless of the physical wire format in which messages are parsed and written. Syntax has been added to ESQL to make it easy to reference element's namespaces using defined prefixes. In Java, XPath expressions are used to reference elements.

Importing from other formats

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

- If you import an XML DTD file, the Message Definition File that is created is 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, the target namespace of the Message Definition File that is created is the target namespace of the XML Schema that is being imported.
  - If namespaces are disabled for the message set, the created Message Definition File is in the notarget namespace. This type of import does not provide full namespace support. If you are using WebSphere Message Broker, you do not have to write namespace-aware ESQL or Java to process an XML message that is parsed against the dictionary that is 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 70
- 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, the target namespace of the Message Definition File that is 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 MRM non-XML messages" on page 36.
  - If namespaces are disabled for the message set, the Message Definition File that is created is in the notarget namespace

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

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

Namespaces with MRM XML messages:

The namespace that is 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 your message set has namespaces enabled, the broker does not store the values of any xmlns 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. When an XML document is written out, the broker regenerates this information from the properties of the XML Wire Format of the message set.

The table of namespace URI/prefix pairs is used by the MRM Domain when it produces 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 attributes that map the prefixes to namespaces. You can choose whether xmlns attributes for all of the entries in the namespace URI/prefix table are written at the start of the document, or whether they are only written 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 map a file name to the notarget namespace. If you are using WebSphere Message Broker, this table is used to produce schemaLocation and No Namespace Schema Location attributes at the start of the XML document.

Namespaces with MRM non-XML messages:

The use of namespaces by WebSphere Message Broker is not necessarily limited to XML message models.

There is one scenario where the use of namespaces by non-XML message models can simplify the ESQL or Java code that you write. But 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 that contains 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 COBOL application requires to be transformed into namespace-aware XML; for example, a SOAP XML message. The transform must map the logical message tree that was 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, 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, this mapping is much simpler.

To enable the MRM parser to create namespace information in a message tree that was 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 cannot do this after 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.

Specifying namespaces in the Message Type property:

When using the MRM domain, the *Message Type* property is used to specify the name of the message.

The format of a simple message type is `{namespace-uri}:name` where *name* is the name of the message, and *namespace-uri.* identifies the namespace. The namespace must be the full URI specification and must be enclosed in braces.

The format `{namespace-uri}name` (that is, with no colon) is also valid. This maintains compatibility with previous versions of the broker product.

If you omit `{namespace-uri}`, the first match for the name that is found in the model is used. You can do this if namespaces are not enabled for the message set, or if a name is unique within a message set. However, if a name is not unique, you must specify the namespace to be sure that the correct match is made in the model.

The following are examples of message types:

- A simple message type for a message in a real target namespace: `{http://www.ibm.com/space}:name`
- A simple message type for a message in the notarget namespace: `{}:name`
- A simple message type for a message in a message set that does not support namespaces: `name`

When identifying an embedded message using a message path, a message type path would be entered as A simple message type for a message in a real target namespace: `{http://www.ibm.com/space}:name`

The same name can occur in more than one namespace. To specify that a name is to be qualified with a specific namespace, the name must be prefixed with the namespace within the Message Type.

For example a Message Type with a single name would be entered as:
`{http://www.ibm.com/space}:id/.../{http://www.ibm.com/space}:name`

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	xsd:import	xsd:include[1]
Parent file has notarget namespace	xsd:import	xsd:include

Note: 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 that includes 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. Therefore, 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 if 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 gives a task list error. A Quick Fix is offered to convert occurrences of xsd:redefine into xsd:include.

Message model integrity

When you create your model, it is important that it is internally consistent and is capable of being generated into the form that you want; for example, a message dictionary or an XML Schema document.

To assist with this, whenever 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 that you have specified for your model have been correctly populated. There is a set of checks for each of the MRM domain physical formats - CWF, XML and TDS. This ensures that the MRM parser can 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 opens the file and positions the editor at the object in error. Organize the task list so that errors are shown before warnings. In this way, errors are not hidden. The task list provides a comprehensive filtering capability if you want 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 because a model that generates 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 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 or the IDOC 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 applications that are 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 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.
- Use the 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.

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.
- 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:xxxxxxxx;B_tag:xxxxxxxx;C_tag:xxxxxxxx}
  ```

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

```
[xxxxxxxx;xxxxxxxx;xxxxxxxx]
```

You supply the precise identification regime as TDS properties.

- A typical XML representation of this model is as follows:

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

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

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

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.

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 physical format layers that you have defined.

Creating physical formats

When 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 see the physical formats in the properties hierarchy pane of the Properties tab. If you click on a physical format for an object, you are presented with a property sheet where you can enter the information for that physical format for that object.

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.

These differences occur because of the different nature of each physical format, and 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.

MRM 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 physical 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: Message 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 *model validation*.

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

Typically, one or other of Length and Length Reference must be set. If Length Reference is set, 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 XML Schema wild cards; 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, because there is no way to recognize the substituted element.

Custom wire format: NULL handling:

CWF supports the handling of explicit NULL values within messages, provided that the logical nillable property of the element is set.

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 that are defined in that message set. The null value of 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. Therefore, if its value is set to 0, for example, both `0` and `0.00` are interpreted as null values.

NullLiteralValue

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

Custom wire format: Multipart messages:

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, you can resolve an embedded message by using ESQL or Java to identify the message. The first message that you reference in this way is assumed to be the selected message. This technique works in an identical manner to unresolved choice handling.

Custom wire format: Data Conversion:

The Custom Wire Format supports the conversion of data to a different code page (for string simple types) or encoding (for 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, this

information can be overridden using the appropriate fields of the WebSphere MQ message header, or other transport header.

Custom wire format: relationship to the logical model:

Some restrictions exist in relation to the logical model for messages that are defined using the CWF.

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

Sequence
You must build the output message to match the sequence of the elements or 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 statement 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 that are built out of sequence contain their default values, not the values that you set. This is because elements that are 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 statement 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 that are built out of sequence contain their default values, not the values that you set. This is because elements that are 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 are written in the order that is 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 are written in the order that is 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 that contains that element.

Message

Mechanisms for the resolution of embedded messages are discussed in the "Custom wire format: Multipart messages" on page 42 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 ESQL 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), inherits 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. These properties 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 produced when writing. Default values are used for missing elements, and any excess elements are discarded.

- 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 writes all occurrences in the message tree, if this number is less than Min Occurs, additional default values are written.

These rules arise because, in a CWF message format, there are no tags or other markup that can be used to determine the end of a variable number of repeats.

However this behavior is overridden if the CWF property Repeat Reference 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.

MRM TDS 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, but it can also handle binary data. Examples of TDS messages are those that

conform to the ACORD AL3, EDIFACT, HL7, SWIFT, or 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; therefore, you can 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 to 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 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*.

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; therefore, 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; therefore, a tag data separator is not necessary.

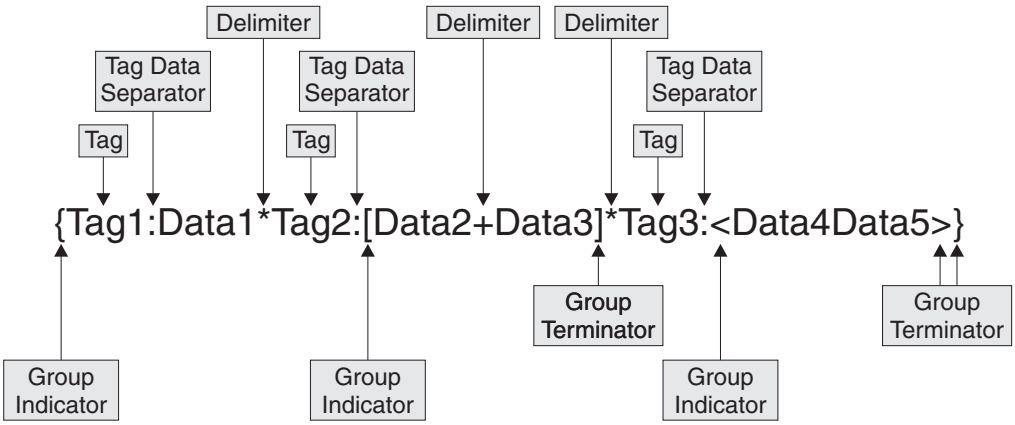
Separation types
The TDS property that controls the way text strings are separated is Data Element Separation. It has several options that let you choose, for example, whether tags are used, whether strings lengths are fixed or variable, and what types of text strings are permitted.

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 (>).

TDS format: Determining the length of simple data values:

The TDS format supports two categories of simple data types: textual and non-textual.

The Physical Type of an element determines whether it is categorized as textual or non-textual.

Textual data
Physical Type is either Text or TLOG Specific. For textual data, the Data Element Separation of the parent complex type or group determines how the length of the data is determined. See "TDS format: Data element separation" on page 47 and its subtopics.

Non-textual data
Elements of all other Physical Types are non-textual. The length of non-textual data is determined by the Physical Type of the element. For non-textual data, the Data Element Separation property of the parent complex type or group does not determine the length, unless Data Element Separation is Use Data Pattern. See "TDS format: Data pattern separation types" on page 56 for more information.

The following table describes how the length of data is determined for each Physical Type.

Physical Type	Determination of Length
Text TLOG Specific	The Data Element Separation of the parent complex type or group determines how the length of the data is determined.
External Decimal Integer Packed Decimal Float Time Seconds Time Milliseconds	Uses the value of the Length property of the element. If Physical Type is Time Seconds, the Length property is set to 4. If Physical Type is Time Milliseconds, the Length property is set to 8. In neither case can this value be changed.
Length Encoded String 1 Length Encoded String 2	Uses the encoded length value in the data.
Null Terminated String	Uses the null terminator at the end of the data.
Binary	Uses the value of the Length Reference or Length property of the element.

TDS format: 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."

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 49.

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 52.

Data pattern types
Data pattern types use a regular expression to identify each element. See "TDS format: Data pattern separation types" on page 56.

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 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 26.

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 textual elements have a length or length reference, and are padded out to their full length in the bit stream. No tags or delimiters are used, and each data value directly follows the preceding data value.

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".

For non-textual elements, the length is determined by the Physical Type of the element. See “TDS format: Determining the length of simple data values” on page 46.

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

For non-textual elements, the length is determined by the Physical Type of the element. See “TDS format: Determining the length of simple data values” on page 46.

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 must be padded out to their full length for the bit stream to be correctly specified to the parser.

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

Default values are required for each field that might not be set, because then every field can be produced as 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 types, but follow extra rules that are 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 bit stream. If a field is missing from the middle of the bit stream, that field is produced for 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 must be padded out to their full length for the bit stream to be correctly specified to a parser.

Tag and delimiter parameters are ignored. Group Indicators and Terminators are observed, because 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 50
- "Tagged Encoded Length separation" on page 51

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. Elements are separated by a predefined delimiter. The textual elements are not of specific lengths. For non-textual elements, the length is determined by the Physical Type of the element. See "TDS format: Determining the length of simple data values" on page 46.

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*tag333:data3}
```

where:

- Group Indicator is {.
- Group Terminator is }.
- Delimiter is *.
- Tag defined for each element, is tag1 (for data1), tag2222222 (for data2), and tag333 (for data3).

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 (for data1), tag22 (for data2), and tag33 (for data3).

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.
- For each textual element, Length or Length Reference indicates the length of the data (this value does *not* include the length of the tag). For non-textual elements, the length is determined by the Physical Type of the element. See "TDS format: Determining the length of simple data values" on page 46.
- Either Tag Data Separator or Tag Length as described above.

Examples: If Tag Data Separator is set to :

```
{tag1:data1tag22222222:data2000tag333:data300}
```

where:

- Group Indicator is {.
- Group Terminator is }.
- Delimiter is *.
- Tag, defined for each element, is tag1 (for data1), tag22222222 (for data2000), and tag333 (for data300).
- Length, defined for each element, is 5 (data1), 8 (data2000), and 7 (data300).

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 (data1), tag22 (data2000), and tag33 (data300).

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.

Only textual elements and elements with a Binary logical and physical type are supported within a Tagged Encoded Length separation.

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

- `tagA007dataAAAtagB006dataBBtagC009dataCCCCC`

  If Length of Tag is 4, Length of Encoded Length is 3, Extra Chars in Encoded Length is 0, then in this bit stream, `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.
- `tagA012dataAAAAAtagB010dataBBBtagC016dataCCCCCCCCC`

  If Length of Tag is 4, Length of Encoded Length is 3, Extra Chars in Encoded Length is 3, then in this bit stream, `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 bit stream 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 bit stream.
- Extra Chars in Encoded Length indicates how many extra characters should be included in calculating the value for the length field in the bit stream.
- Either Tag Data Separator or Tag Length as described above.

Examples: If Tag Data Separator is set to :

```
{tag1111:008data1tag222222222:010data2AAtag3333:009data3A}
```

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, tag222222222, 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 bit stream and elements cannot be omitted from the middle of the bit stream.

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 bit stream.
- 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" on page 53
- "Variable Length Elements Delimited" on page 54

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, 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 property determines whether this is done. If this property is set to End of Type, this can be done (with one exception, shown below).

  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 the Suppress Absent Element Delimiter property is set to Never, 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 where the same delimiters are 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, with all elements present, you might have:

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

If `element3Data2` and `element3Data3` are missing, and the delimiters are suppressed, it is not possible for the parser to determine which elements are missing.

Therefore, in this case, you must override the Suppress Absent Element Delimiter property, and write out all the delimiters to clearly define the message to the parser. Therefore, the string 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 bit stream is not clear to the parser.

All Elements Delimited: In an All Elements Delimited separation type, all elements are separated by a delimiter; for example:

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

where Delimiter is *.

An All Elements Delimited separation type does not use tags or their associated parameters.

For textual elements, the length is determined by the delimiter, and the Length property is ignored unless the Observe Element Length property is set.

For non-textual elements, the length is determined by the Physical Type of the element. See "TDS format: Determining the length of simple data values" on page 46.

Applicable properties: These properties 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 indicates the separator between the data elements within a group or complex type.
- Suppress Absent Element Delimiters indicates whether delimiter suppression is permitted (see below).

For example:

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

where:

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

Repeating element rules: If an element needs to be repeated when the separation type is All Elements Delimited, 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 property is set to End of Type, you can use delimiter suppression. Therefore, if only the first `data2` element was present in the previous example, the bit stream reads:

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

However, if the Suppress Absent Element Delimiters property is set to Never, the bit stream reads:

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

If Delimiter and RED match, two delimiters are output to indicate that the repeat is ending. Therefore, if the delimiter and RED are *, the bit stream reads:

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

Variable Length Elements Delimited: In a complex type with Variable Length Elements Delimited separation, some elements are determined by their length, and other elements are delimited. This combination of a delimited and a fixed length format follows rules that are associated with both formats. Lengths can be given and used, but they are not mandatory.

- If a length is present for a textual element, 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 for a textual element, the delimiter is required.
- For non-textual elements, the length is determined by the Physical Type of the element. See "TDS format: Determining the length of simple data values" on page 46.

A complex type with Variable Length Elements Delimited separation that contains only variable length elements resembles a acomplex type with All Elements Delimited separation. 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 properties: The following properties 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 indicates the separator between the data elements within a group or complex type.
- Suppress Absent Element Delimiters indicates whether delimiter suppression is permitted.
- (Optionally) Length or Length Reference indicates the length of a textual element. If a textual element has a length, this length is used. Because the length

of this element is known, it is not necessary to output a delimiter after it. If the length is not known, a delimiter is required. A delimiter is never required for a non-textual element.

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 Length 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, 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, you can use delimiter suppression.

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

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

However, if Suppress Absent Element Delimiters is set to Never, the bit stream reads:

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

If the delimiter and RED match, two delimiters are output to indicate that the repeat is ending. So if the delimiter and RED are both *, the bit stream reads:

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

This also applies for a non-fixed length complex type or group inside a Variable Length Elements Delimited environment.

Fixed Length Element Repeating: If an element with a defined length (a fixed length element) is repeating and the minimum occurrences is not the same as maximum occurrences, an RED is not required, but a delimiter *is* required after the last repeat. Delimiter suppression of this repeat can occur.

For example, if `data4` (with a fixed length of 8) is repeating, and its minimum occurrences is 2, maximum occurrences is 4:

```
data1*data2*data3*data400data400data400data400*data5
```

where Delimiter is *

Or, if there are only two occurrences of `data4`:

```
data1*data2*data3*data4000data4000*data5
```

If an element with a defined length (a fixed length element) is repeated, and the minimum occurrences is the same as maximum occurrences, an RED is not required. A delimiter is also not required after the last repeat. Truncation of this repeat cannot occur and all elements need to be present.

For example, if `data4` (with a fixed length of 8) repeats four times:

```
data1*data2*data3*data4000data4000data4000data4000data5
```

where Delimiter is *

Or, if there are only two occurrences of `data4`:

```
data1*data2*data3*data4000data400000000000000000000data5
```

This also applies for a non-fixed length complex type or group inside a Variable Length Elements Delimited environment.

If a complex type has Variable Length Elements Delimited separation, a delimiter is always output between an included ('child') complex element and the next element even if the separation of the 'child' complex element is Fixed Length. On input, the parser accepts the bit stream with or without such a delimiter.

TDS format: Data pattern separation types:

For a data pattern separation type, each data value is matched with a regular expression that is specified as a property of each element.

The length of both textual and non-textual data is determined by the Data Pattern property of the element. If the Physical Type of the element is Length Encoded String 1 or Length Encoded String 2, the regular expression must match both the encoded length and the following data. The length in the encoded length must be consistent with the length matched by the regular expression. If the Physical Type of the element is Null Terminated String, the regular expression must match both the data and the following null terminator.

The Data Pattern separation type uses a regular expression that is specified for each element to match the data. The parser matches the data with the regular expression in the Data Pattern property for that element. TDS parsing in the MRM parser uses the regular expression in Data Pattern to determine the length of the element, whether it is repeating, and whether it is present in the bit stream.

No delimiters or tags, other than those coded as part of the regular expression pattern, are used in the bit stream. See “Using regular expressions to parse data elements” on page 776 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.

If the TDS message that is being parsed is encoded in a single-byte code page, the Data Pattern property can include hexadecimal values. A hexidecimal value is specified as \xNN, where N is a hexadecimal digit in the range 0 to F. Note, however, that the value \x00 is not valid.

Performance issues

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

Therefore, use Data Pattern separation type only when no other separation type models the message. Do not use it, for example, when you can use Fixed Length separation type.

Applicable parameters: Only one parameter is used:

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

TDS format: 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. 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 restrictions follow the rules specified in "TDS message model integrity" on page 772. Most rules are applied for at least one of these reasons:

Rules for message definition

Some rules are necessary for the message to be 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 bit stream 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 is that it is not possible to have a Tagged Delimited separation type inside a Fixed Length type. Because 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 rules occur where a group composition or group content validation cannot be used 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 bit stream.

TDS format: NULL handling:

NULL handling dictates the way in which the MRM parser for TDS messages handles elements that have been set to Null.

Null handling only takes place if the logical Nillable property of the element is set. The rules for whether nulls are permitted are described in "TDS Null handling options" on page 770.

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 enumerated values NULLPadFill, NULLLogicalValue, NULLLiteralValue, and NULLLiteralFill. 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.

TDS 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 by 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 might 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.

TDS format: Data conversion:

TDS string data is subject only to CCSID 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.

TDS format: Relationship to the logical model:

TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

The rules that govern these options are explained in "Restrictions for nesting complex types" on page 774.

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.

However, this behavior is overridden if the TDS 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. See "Repeat reference" on page 61 for more information.

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	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. • 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.
All Elements Delimited	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. A varying number of occurrences (Min Occurs <> Max Occurs) is allowed if Suppress Absent Element Delimiters is set to End of Type. • If the Delimiter is different from the Repeating Element Delimiter, the Delimiter will signify the end of the occurrences. • If the Delimiter is the same as the Repeating Element Delimiter, an empty repeat signifies the end of the occurrences. • In both these cases, Max Occurs is the maximum number of repeats that are expected. If Suppress Absent Element Delimiters is Never, all occurrences are expected when parsing, and produced when writing, although parsing will accept elements being absent. Optional occurrence (Min Occurs = 0) is ignored and a delimiter is still expected when parsing, and produced when writing. Always absent (Max Occurs = 0) is allowed. No delimiter is expected when parsing, nor output when writing. 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 bit stream. On writing, the writer outputs all occurrences in the message tree.

Separation type	Interpretation of Min Occurs and Max Occurs
Fixed Length Fixed Length AL3	Max Occurs only is used when parsing and writing. In general, Max Occurs occurrences are expected when parsing, and Max Occurs occurrences are produced when writing; default values are used for missing elements, and any excess elements are discarded. 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. 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, additional default values are written.
Variable Length Elements Delimited	For fixed length simple elements, the rules for Fixed Length separation above are followed with two differences. 1. A varying number of occurrences (Min Occurs <> Max Occurs) is allowed, the end of the occurrences being signified by an extra delimiter. 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. For variable length simple elements, all complex elements and groups, the rules for All Elements Delimited above are followed.
Data Pattern	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. 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.

Repeat reference

The TDS property Repeat reference specifies a field that holds the number of repeats of an object (Element or Group) within a message. The field that holds the number of repeats must be within the message before the object that it refers to.

From a parsing perspective, the Repeat reference property replaces the role of the minOccurs and maxOccurs properties.

If a value for the Repeat reference property is specified for an object, values that are specified for minOccurs and maxOccurs are ignored when parsing and writing. However, values that are specified for minOccurs and maxOccurs are used by logical validation.

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

Separation type	Interpretation of Repeat reference
Tagged Delimited Tagged Fixed Length Tagged Encoded Length	Repeat reference is 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.
All Elements Delimited	Repeat reference 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. A Repeat reference is allowed only if the parent complex type or group has Suppress Absent Element Delimiters set to Never. All Repeat reference occurrences are expected when parsing, and produced when writing. However, parsing accepts elements being absent. Repeat reference = 0 is allowed. No delimiter is expected when parsing, nor produced when writing.
Fixed Length Fixed Length AL3	Repeat reference is used when parsing and writing. Repeat reference occurrences are expected when parsing, and are produced when writing, with default values used for missing elements. Repeat reference = 0 is allowed.
Variable Length Elements Delimited	For fixed length simple elements, the rules for Fixed Length separation above are followed. For variable length simple elements, all complex elements and groups, the rules for All Elements Delimited that are listed above are followed.
Data Pattern	Repeat reference is 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.

MRM XML physical format

The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

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. The physical format name identifies the definitions that are to be used by the message broker at run time.

After adding an XML physical format, all XML properties for all existing objects in the message set are set to default values. Therefore, immediately after adding the format and deploying the message set to a runtime environment, you can 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 element name in the model.

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 that are not in the definition for that message.

- If such an element exists in the message set, the model 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 'by hand', 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: Message 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 *model 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.

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.

MRM XML physical 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 the MRM XML physical format 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.

Null handling only takes place if the logical Nillable property of the element is set.

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>
```

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

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

Or, the element could follow XML Schema instance rules:

```
<element1 xsi:nil="true"/>
```

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`.

MRM XML physical format: Multipart messages:

Identify embedded messages by using either a Message Identity or a Message Path.

If you are using the MRM XML physical format, an embedded message can be identified in any of the following ways:

- Message Identity

  See "Identifying an embedded message using a Message Identity" on page 27.
- Message Path

  See "Identifying an embedded message using a Message Path" on page 29.
- Automatic

  The MRM parser identifies the message by matching the next XML tag in the bit stream against the XML Name of a message definition.

If you choose the Message Identity or Message Path technique, the MRM parser still checks that the next XML tag name matches the XML Name of the message that was identified. If the XML Name does not match, an exception is thrown.

Where you have defined the embedded message in a different message set, you need to use a Message Set Identity element or attribute value to specify the target message set.Note that the message sets within which the root and subsequent embedded messages are defined must be consistent in their use of the 'Use Namespace' property of the message set. That is, embedded messages that are defined in a namespace-aware message set and that are contained within a parent message that is defined in a message set that is not namespace-aware, are not supported. Similarly, embedded messages that are defined in a message set that is not namespace-aware and that are contained within a parent message that is defined in a namespace-aware message set, are not supported.

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 bit stream 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.

MRM XML physical format: relationship to the logical model:

The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

These restrictions are documented in “MRM restrictions” on page 747.

Default values

The MRM XML physical 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 validating the content of a message.

When parsing and writing, using the MRM XML physical format, 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.

MRM XML physical format: Handling xsi:type attributes:

The prefix "xsi" is the namespace prefix used by convention for the XML Schema instance namespace. XML documents can contain elements that have an xsi:type attribute. This behavior provides an explicit data type for the element.

The MRM XML parser in 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 produces 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 produced 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, the behavior 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, 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. 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, an xsi:type attribute can be used to direct the MRM XML parser when resolving the union.
- If you have migrated from an earlier version of WebSphere Message Broker that was not sensitive to xsi:type attributes, you might notice some changes of behavior. For example, 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, change your message flow to comply with the current 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 instance namespace and XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

Ways to create message definitions

When you have created a message set, you must populate the message set with message definitions.

You can populate the message set in one of the following ways:

- By importing application message formats that are described by XML Schemas, IBM supplied messages, XML DTD, C structures, COBOL structures, or WSDL definitions; use the wizards that are started from the New Message Definition File From wizard.
- By creating empty message definition files, then creating your messages by using the Message Definition Editor; use the New Message Definition File wizard.
- By using the Adapter Connection wizard to import EIS metadata.

Additionally, you can import application message formats by using the mqsicreatemsgdefs or mqsicreatemsgdefsfromwsdl command line utilities.

The mqsicreatemsgdefs command has a bulk import capability, but mqsicreatemsgdefsfromwsdl imports only one WSDL definition at a time.

Importing from other model representations to create message definitions

You can add message definitions to your message set by importing application message formats that already exist.

You can import the following message formats into your message set:

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

When you import one of these formats, a new message model is created that consists of the elements, attributes, groups, and types that are required to represent your message format. You can 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 that is created can consist of both logical and physical information, if appropriate physical formats exist in the message set at the time of the import.

To find out which wizards to use to import message formats, see "Ways to create message definitions" on page 66.

You can also import C header files, COBOL copybooks, XML DTD files, or XML Schema files by using the mqsicreatemsgdefs command line utility. The mqsicreatemsgdefs command allows you to import several message format files in a single operation, and allows you to create a message set (based on an existing message set) into which the message definition files are placed.

WSDL definitions can be imported by using the mqsicreatemsgdefsfromwsdl command line utility. This utility imports only one WSDL definition at a time.

Client application access to messages: Client applications must have access to message definitions to be able to construct the messages that they send, and to interpret the messages that they receive.

- If the message formats have been imported from C or COBOL structures by 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 is used by the brokers.
- If the messages are self-defining XML, the client applications must construct valid messages by using the structures that can be understood by the recipients of the message.

Importing from XML Schemas to create message definitions:

You can populate a message set with message definitions by importing XML Schema files, using the **New Message Definition File From XML Schema file** wizard, the **Start from WSDL and/or XSD files** quick start wizard, or the mqsicreatemsgdefs command line utility.

Each XML Schema file that 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 From XML Schema 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 that is 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 therefore resides 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 70.

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 From XML Schema file wizard: When you import using the **New Message Definition File From XML Schema file** wizard, you can specify which of the global elements or global complex types 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 into the same message set using a separate import operation.

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 and global complex type 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, 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 132 for details.

MRM 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 are 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 do not prevent your model being generated in another form; for example, 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 must be 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>` that holds a set of information that applies to all XML Schema files that are imported during an invocation of the utility. A default XML options file, called `mqsicreatemsgdefs.xml`, is supplied. If you want to apply different sets of information to different XML Schema files, you must create multiple XML files and run the utility multiple times.

When you are importing into a message set for which namespaces are not enabled, there are two other options that you can specify in the `<XSD_NO_NS>` element in the XML options file:

- 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 that contains a XML physical format and pre-populated message set level XML properties, which are then copied into the message set that is 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.

When you import 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 XML `no target` namespace. In some cases, this action 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 conflicts cause error entries in the task list that you must resolve before generating the model in another format, such as a message dictionary.

Because all the message definition files are in the XML `no target` 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 imported XML Schema file must specify an xmlns attribute with a non-empty prefix for the target namespace of the XML Schema file. This prefix is used in the XML names in the XML Wire Format layers.

Therefore 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 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 mqsicreatemsgdefs 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.

Importing 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 From IBM supplied messages** wizard.

Each IBM supplied message that you import results in a new message definition file within 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 From IBM supplied messages** wizard allows you to choose a different file name.

See "Importing from the command line" on page 132 for information about what IBM supplied messages can be imported.

When you import using the **New Message Definition File From IBM supplied messages** wizard, you can specify only 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 that allows you to examine what occurred during the import process and check any errors that resulted.

Importing 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 From XML DTD file** wizard or the mqsicreatemsgdefs command line utility.

Each XML DTD file that 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 From XML DTD file** wizard allows you to choose a different root file name.

If the message definition file 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 that are created as a result of DTD file import belong to the `noTarget` XML namespace and so reside in the `(default)` location in your workspace.

A report file is created during the import operation, 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 From XML DTD file wizard: When you import using the **New Message Definition File From XML DTD file** wizard, you can specify which of the elements within the imported XML DTD file are to be messages within the message definition file.

You can import only one XML DTD file with each import operation.

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 element within the imported XML DTD file. The import operation creates a message and a corresponding element in the message definition file for each element that 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, 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 132 for details.

MRM CWF and TDS physical format properties are *not* populated and therefore 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 that you have imported are 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 do 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 must be passed as a parameter to the utility.

The mqsicreatemsgdefs utility also allows you to create a 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 that contains a XML physical format and pre-populated message set level XML properties, which are then copied into the message set that is created by the import.

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

Importing 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 From C header file** wizard or the mqsicreatemsgdefs command line utility.

Each C header file that 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 From C header 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 that are created as a result of an import from a C header file belong to the `noTarget` XML namespace and therefore reside in the `(default)` location in your workspace. This default namespace can be overridden by specifying a target namespace. See "Namespaces with MRM non-XML messages" on page 36 for reasons why you might want to do this.

In your C header file there are typically one or more C structures. You can select which of these structures to import. The import operation then imports those structures, plus any others that 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 a corresponding global element in the message definition file for each structure that 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.

If you import using the **New Message Definition File From C header file** wizard you can import only one C header file with each import operation. But, if you import 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 for a successful compilation, 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, 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.

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, 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 must 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.

Importing 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 From COBOL file** wizard or the mqsicreatemsgdefs command line utility.

Each COBOL copybook that 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 From COBOL 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 that are created as a result of COBOL copybook file import belong to the noTarget XML namespace and therefore reside in the (default) location in your workspace. This default namespace can be overridden by specifying a target namespace. See "Namespaces with MRM non-XML messages" on page 36 for reasons why you might want to do this.

In your COBOL copybook file there are typically 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 that 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 that 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.

If you import using the **New Message Definition File From COBOL file** wizard, you can only import one COBOL copybook file with each import operation. If you use 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, 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. 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 132 for details.

XML and TDS physical format properties are *not* populated and therefore 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 that 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 are set accordingly. There is a range of environments supported; if your environment is not shown, 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>` that holds one set of environment specific information that applies to all COBOL copybook files that are 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 must 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 that contains a CWF physical format and pre-populated message set level CWF properties, which are then copied into the message set that is created by the import.

Importing from WSDL files to create message definitions:

Import WSDL files, using the **New Message Definition File From WSDL file** wizard, the **Start from WSDL and/or XSD files** Quick Start wizard, or the mqsicreatemsgdefsfromwsdl command.

Each WSDL file that you import results in one or more new message definition files within the message set. A new message definition file is created for each namespace that is 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 From WSDL file** wizard allows you to choose a different file name.

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

The message set that you are importing the WSDL file into must be namespace-enabled. If it uses the MRM domain, it must have an XML physical format so that the message set is suitable for the runtime parsing of XML messages such as SOAP.

Use the report generated during the import operation to see what happened and to check any errors.

You specify a single WSDL definition for each import operation. If the WSDL definition consists of a hierarchy of files, you must supply the name of the file that contains the WSDL service or binding definitions. The WSDL definition that is being imported must contain one or more WSDL bindings for the import to proceed.

Importing using the New Message Definition File wizard

When you import using the New Message Definition File wizard, you can specify only one WSDL definition for each import operation. A WSDL definition could be held as one or more WSDL files, which are all imported as a result of importing the definition. The WSDL definition being imported must contain one or more WSDL bindings for the import to proceed.

Importing using the command line

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

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`. They are not removed after the import; the user can later update or run validation on them in the workbench.

Physical information

An XML physical format layer is required for the MRM domain, and must be added to an existing message set prior to importing the WSDL definition.

Relationship of WSDL to Message Model:

If a broker is to communicate with an existing Web service, it typically needs to send and receive SOAP messages. To take this approach, use the MRM domain. You must ensure that the broker message model and the WSDL definition used by the Web service describe the same messages. In general, you can achieve this result by importing the WSDL for the existing Web service by using the broker tooling. Currently only WSDL version 1.1 is supported.

Only the WSDL operation, message, and part definitions are represented in the resulting broker model. Starting with the lowest level, a WSDL definition describes the following resources:

- A number of logical messages and their constituent parts, which are 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 no targetNamespace is defined on the <xsd:schema> element, the resulting elements and types have no namespace.
- A number of operations that form the WSDL portType or interface. The operations define the business payload for the SOAP messages at run time. Broker messages are created for each possible payload. In the case of rpc-style WSDL, the message model needs message definitions that correspond to the WSDL operations themselves. The names of these message definitions are derived from the WSDL operation name, and their namespace is given by the namespace attribute on the WSDL <soap:body> definition.
- A SOAP version and encoding style. Message definitions for the SOAP envelope and, if necessary, the SOAP encoding scheme are imported into the message set. Currently the WSDL importer assumes the use of SOAP version 1.1. WSDL version 1.1 can define a Web service that uses SOAP version 1.2, but no standard method exists to achieve this definition. If your Web service does use SOAP version 1.2, you might have to remove the SOAP version 1.1 definitions manually and import the SOAP version 1.2 definitions.

Resulting message model

The resulting model allows the user to parse incoming SOAP messages by using the MRM XML parser where the message type is 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 that are created by the WSDL importer to appear at these points.

The allowed attachment points are Envelope.Body, Envelope.Header, and Envelope.Body.Fault.detail. A message from the user's message model might appear at each point (in the case of the Envelope.Header, multiple messages might appear). In the case of rpc-style WSDL, the message expected at Envelope.Body is the automatically-generated message that corresponds to the WSDL operation. In all other cases, the messages expected are those defined by the WSDL message parts for each operation.

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 a broker, a parser, or your applications.

The following representations are supported:

- A message dictionary, for deployment to a broker.
- A W3C XML Schema, for use by an application building or processing XML messages, or for deployment to a broker.
- Web Services Description Language (WSDL), for a Web services client application, or for deployment to a broker.
- Documentation, to give to programmers or business analysts.

Generation for deployment to a broker takes place automatically when you add your message set to a Broker Archive (BAR) file.

Generation for other purposes is achieved using the **Generate** menu actions.

Generate message dictionaries

A *message dictionary* is data structure that describes all of the messages in a message set in a form suitable for deployment to the MRM parser.

Purpose of a message dictionary: A dictionary describes the logical structure and content of a set of messages, and typically contains one or more physical formats that describe how those messages are serialized in a bit stream. The MRM parser within WebSphere Message Broker uses this information to parse an incoming message bit stream into a message tree, or to write a message tree into a physical bit stream.

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 that the MRM parser 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 that 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. To do this, add the message set to a BAR file, then deploy 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, if the message set supports the MRM domain.

Before adding a message set to a BAR file, the Message Broker 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 message dictionary is generated.

Dictionary generation report files:

Whenever a message dictionary is generated, a user log file is also generated and added to the same BAR file. This file contains informational messages and warnings that relate to the use of the generated dictionary. Always check this file after generating a message dictionary.

Generate XML schema

Generate a schema file from a message definition file.

XML schema is a standard way of describing complex message models.

You can generate a schema file for an individual message definition file, or for all message definition files in a message set. If any XML physical formats have been defined for the message set, you can select which of these XML wire formats are to be applied.

- If an XML format has been selected, the physical format information is also 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.

Validating an XML message against a schema:

The XMLNSC parser can validate an XML message against any deployed XML schema.

You can validate an XML message against an XML schema when the message is parsed, when the message is serialized, or at any point within a message flow.

To construct a message flow for schema validation, you must perform the following tasks:

1. Enable validation at the appropriate point within the message flow. See Validating messages.
2. Ensure that you have deployed all XML Schema files that are required. See "Deploying an XML Schema."
3. Specify the message set in which the schema was deployed; this is done using the MessageSet property of the message. See Accessing the Properties tree.

Schemas are deployed within a message set, and are identified by supplying the message set name in the message properties.

Deploying an XML Schema:

XML Schemas are created as Message Definition Files within a message set that is then deployed.

To create and deploy a message set for schema validation you must:

1. Create or locate a message set that will contain the schemas.

2. Set the Message Domain property of the message set to XMLNSC.
3. Use the New Message Definition File wizard to create a message definition file (mxsd) from the XML Schema file (.xsd).
4. Add the message set to a BAR file and deploy the BAR file.

Repeat step 3 for each XML Schema file that you want to deploy.

If your XML Schema imports or includes other XML Schema files, you can use the mqsicreatemsgdefs command to create all the message definition files in a single operation.

Generate WSDL

A *Web Services Description Language* (WSDL) document specifies the interface to a Web service, and enables a Web service client to invoke it. A WSDL document that is generated from a message set defines Web service requests and responses in terms of the messages that you have defined in that message set.

Use message definition files with target namespaces when you generate WSDL. If you do not, WebSphere Message Broker defaults the target namespace to the WSDL target namespace.

If the WSDL uses a message from the message definition file, one XML Schema file is generated for each message definition file in the message set. Within the main WSDL document, operations are defined in terms of logical messages, which are themselves defined in terms of the elements and types that are defined in these XML Schema files.

WSDL operations are grouped into a logical interface or portType, and are then associated with a binding which defines the physical format of the messages. You can select only one of the following bindings when you generate WSDL:
- SOAP (over JMS)
- SOAP (over HTTP)

A WSDL service definition specifies the endpoint where the service is available. You can elect to have the service, binding, and portType definitions generated as a single file or as separate files, but the XML Schema files are always generated separately. Tools that consume WSDL are typically more tolerant of the single-file format.

Relationship to the message model when generating WSDL:

If a broker is to communicate with a Web service client, it must typically accept SOAP messages. You can model your messages in the MRM domain, and the message model you deploy to the broker 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), you can export the model to create a corresponding WSDL definition for use by the client. At the same time, you must enhance your message model with appropriate definitions for the SOAP envelope and (for rpc-style) the WSDL operations. Currently only WSDL version 1.1 is supported.

To generate WSDL, you need:

1. A way of representing the WSDL operations. The message category performs this service.
2. Away of representing the data for these operations. The message model performs this service
3. A way of soliciting the Web service end-point and binding details. The WSDL Generator wizard performs this service.

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 run time, the format of the SOAP messages depends on the WSDL style specified in the wizard. If you select rpc-style, the SOAP Envelope contains a message that corresponds to a WSDL operation. The WSDL generator adds an appropriate message definition that corresponds to the WSDL operation to your message set. If you select document-style, the SOAP envelope contains messages specified in the category, therefore you do not have to add additional message definitions 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 incoming SOAP messages to be parsed by 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 supported attachment points are Envelope.Body, Envelope.Header and Envelope.Body.Fault.detail. A message from your message model might appear at each point (in the case of the Envelope.Header, multiple messages can appear). For rpc-style WSDL, the message expected at Envelope.Body is the automatically generated message that corresponds to the WSDL operation; for example, the message category. In all other cases, the messages expected are those referenced by the message categories.

Generating message set documentation

Message set documentation describes, in a human-readable format, message definitions that 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

Creating and deleting a message set.

Before you begin to develop your message model, you must create a message set. A message set project is automatically created when you create a message set.

This topic area describes the tasks that are involved in working with a message set.

- "Creating a message set" on page 83
- "Deleting a message set project"

Deleting a message set project

Delete a message set project and, optionally, the contents of the associated project directory.

Before you start:

You must have completed the following task:

- "Creating a message set" on page 83

Tip: Close all open windows within the workbench that relate to the message set project or associated files that you want to delete. If you do not do this, errors might occur when you try to process objects that no longer exist your workspace.

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 Broker Development view, 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. To delete the contents of the project directory, click **Also delete contents**; all files and directories that are 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

Complete a variety of tasks that are involved in working with a message set.

- "Configuring message set preferences" on page 83
- "Opening an existing message set" on page 83
- "Creating a message set" on page 83
- "Configuring logical properties: Message sets" on page 86
- "Working with physical formats" on page 87
- "Configuring documentation properties: Message sets" on page 93
- "Deleting a message set" on page 93
- "Applying a Quick Fix to a task list error" on page 93

Configuring message set preferences

This task topic explains how to make changes to preferences that relate to message set processing. These preferences are for message set editors, message set model 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 **Broker Development > Message Sets** 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

Open an existing message set in the Message Set editor so that you can view or edit its contents.

Before you start:

Create a message set by following the instructions in "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 Broker Development view, right-click the `messageSet.mset` file of the message set that you are opening then click **Open** on the pop-up menu. This opens the Message Set editor for the selected file.

You can now view or edit the file as required.

Creating a message set

Use the New Message Set wizard to create a message set.

The New Message Set wizard also creates a new message set project.

Note: You can also use a Quick Start wizard to create a message set, a message set project, and other resource files that you need to create a new application.

The New Message Set wizard allows you to select what kinds of message format you want to model in your message set. The message domain and the physical format created is inferred from the selection that you make. Note, however, that you can change the inferred domain using the message set editor.

The options are:

- XML documents (general)
- Web services (SOAP)
- Binary data (for example, C or COBOL structures)
- Text data (for example, CSV, SWIFT, or HL7)
- MIME documents other than Web services
- Data for WebSphere Adapters

The default value is `XML documents (general)`.

Below the list of message formats there are check boxes corresponding to each of the message formats. The check box corresponding to the message format that you selected is not available, but you can select any of the other check boxes to add other message formats to your message set.

If you later select a different default message domain, the checked state for the domain that you originally selected as the default does not change, but the check box is enabled.

As you can now select more than one message domain you can, for example, use the default value of **XML documents (general)** together with **Binary data (for example, C or COBOL structures)** and **Text data (for example, CSV, SWIFT or HL7)**. This results in the selection of the XMLNSC and MRM domains (to handle non-XML documents) within the same message set if you require this functionality.

The mapping between the selection, the domain, and the wire format created is described in the following table:

Selection	Inferred message domain	Physical format created
XML documents (general)	XMLNSC	XML
Web services (SOAP)	SOAP and XMLNSC	XML
Binary data (for example, C or COBOL structures)	MRM	CWF
Text data (for example, CSV, SWIFT, or HL7)	MRM	TDS
MIME documents other than Web services	MIME	None
Data for WebSphere Adapters	DataObject	None

Depending on your selection, an appropriate IBM supplied message will be imported into the message set.

Note: The XML physical format is created only in case the user switches to MRM XML.

If you click **Finish** on the second page of the New Message Set wizard, the message set that is created has the following default property values:

Property	Default value
Message Domain	XML documents (general)

Property	Default value
Physical Format	XML Wire Format (XML1)
Namespace support	Enabled

To create a new message set:

1. Switch to the Broker Application Development perspective.
2. Open the New Message Set wizard. To do this, right-click anywhere in the Broker Development view, then click **New → Message Set**.
3. In the **Message set name** field, type the name for the message set that you are creating. The name that you type is also displayed in the **Message set project name** field.
4. Optional: You can choose a different message set project name; to do this, type this name into the **Message set project name** field.
5. Optional: You can specify a directory in which you want to store the project contents. If you do not specify a directory, the default workspace is used. To specify a directory, first clear the **Use default** check box, then either type into the **Directory** field the location of the directory, or click **Browse** to see a list of the folders that you can choose from for the location of the directory.
6. Optional: If you want to create a new message set whose definition is based on existing message set, click **Message Set** in the **Copy message set contents from another message set** pane and choose from the list of message set definitions that are shown; then click **Finish**. The new message set (and the message set project that contains it) is created immediately and the New Message Set wizard automatically closes.
7. Optional: If you want to create a message set whose definition is not based on an existing message set, click **Next**. You are presented with the next pane which allows you to choose the type of message data that you want to process.
   a. Expand the list shown under **Select the type of message data that you will be working with most often** and choose a value from the list shown. The value that you choose determines the default message domain of the message set. If you choose `XML Documents (general)`, the default message domain XMLNSC is used.
   b. Optional: You can now select additional types of message data. Under **Select any other types of message data that you will be working with** there are check boxes for each of the following message data types:
      - XML documents (general)
      - Web services SOAP
      - Binary data (for example, C or COBOL structures)
      - Text data (for example, CSV, SWIFT or HL7)
      - MIME documents other than Web services
      - Data for WebSphere Adapters

      Note: These check boxes correspond to the list of data types from which you chose the data type that you will be working with most often, but the check box that corresponds to the data type that you chose from that list is not available.

      By default, all these check boxes are cleared. You can select any, or all of these check boxes, to add the corresponding data types to your message set.

      If you select the check box for text data, either for the type of message data that you will be working with most often or as another type of message

data that you will be working with, you can additionally choose from the displayed list of text messaging standards. This list is the same as that given in the description of the Messaging Standard property in "TDS Format message set properties" on page 162.

c. Click **Next**. A new panel is displayed that summarizes some information about the message set that you have created. Specifically, it lists:

   Supported message domains

   Physical formats to be created

   IBM supplied messages to be imported

8. Click **Finish** on this page to create the message set, and the message set project that contains it. The New Message Set wizard closes.

After the New Message Set wizard finishes, the message set editor is opened.

You can now create some message definitions in the new message set. You can either create new message definitions from scratch, or create them based on existing artifacts such as WSDL, XSD, DTD, C, COBOL files, or EIS metadata. Use the Message Definition File wizard and the Message Definition File From wizard to help you with this.

Configuring logical properties: Message sets

Configure the logical properties of a message set using the Message Set editor.

Before you start:

You must have completed the following task:

- "Creating a message set" on page 83

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 83.

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 to your requirements the logical properties that are shown in the Details view.

   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; the **Message Set ID** must not then be altered.
4. Save your changes by clicking **File> Save** or by pressing Ctrl+S. Alternatively click **File> Save All** or press Ctrl+Shift+S.

Working with physical formats

If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Opening an existing message set" on page 83

This topic area covers the following tasks that relate 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 88
- "Adding a TDS physical format" on page 88
- "Configuring TDS properties: Message sets" on page 89
- "Adding an XML wire format" on page 89
- "Configuring XML Wire Format properties: Message sets" on page 90
- "Renaming a physical format" on page 90
- "Applying default physical format settings: Message sets" on page 91
- "Removing a physical format" on page 91
- "Observing 2007 U.S. changes to daylight saving time" on page 92

Adding a Custom Wire Format (CWF)

You can add a Custom Wire Format (CWF) physical format layer to a message set by using the Message Set editor.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Opening an existing message set" on page 83

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 **Custom Wire Formats**, then clicking **Add Custom Wire Format**.
3. On the Add Custom Wire Format window, specify the name that you want to use for the new CWF physical format. The default name is 'Binary1'.

   Tip: Physical format names must be unique across a message set. You are recommended to start the name of your new CWF physical format with 'CWF' or 'Binary', because this clearly identifies the type of the physical format that you are adding in relation to any of the other types.
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** or by pressing Ctrl+S.

Configuring Custom Wire Format (CWF) properties: Message sets

Configure the Custom Wire Format (CWF) properties of a message set using the Message Set editor.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Opening an existing message set" on page 83
- "Adding a Custom Wire Format (CWF)" on page 87

To configure the CWF properties of a message set:

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, the **Custom Wire Formats** node of the Properties Hierarchy shows the name of each of the CWF physical formats that have been added to the message set. If the physical format names are not in view, expand **Custom Wire Formats** by clicking **+**.
3. Click the chosen CWF physical format so that the properties of this format appear in the Details view of the Message Set editor.
4. Configure the CWF properties shown in the Details view according to your requirements.
5. Save your changes either by clicking **File> Save** or by pressing Ctrl+S. Alternatively click **File> Save All** or press Ctrl+Shift+S.

Adding a TDS physical format

Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Opening an existing message set" on page 83

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 **Tagged/Delimited String Formats** 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. The default name is 'Text1'.

Tip: Physical format names must be unique across a message set. You are recommended to start the name of your new TDS physical format with 'TDS' or 'Text', because this clearly identifies the type of the physical format that you are adding in relation to any of the other types.

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** or by pressing Ctrl+S.

Configuring TDS properties: Message sets

Configure the Tagged/Delimited String (TDS) format properties of a message set using the Message Set editor.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Opening an existing message set" on page 83
- "Adding a TDS physical format" on page 88

To configure the *TDS* physical format properties of a message set, do the following:

1. Switch to the Broker Application Development perspective.
2. In the Message Set editor, the **Tagged/Delimited String Formats** node of the Properties Hierarchy shows the name of each of the TDS physical formats that have been added to the message set. If the physical format names are not in view, expand **Tagged/Delimited String Formats** by clicking **+**.
3. Click the chosen TDS physical format so that the properties of this format appear in the Details view of the Message Set editor.
4. Configure the TDS properties shown in the Details view according to your requirements.
5. Save your changes either by clicking **File> Save** or by pressing Ctrl+S. Alternatively click **File> Save All** or press Ctrl+Shift+S.

Adding an XML wire format

You can add an XML wire format physical format layer to a message set by using the Message Set editor.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Opening an existing message set" on page 83

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 **XML Wire Formats**, 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. The default name is 'XML1'.

   Tip: Physical format names must be unique across a message set. You are recommended to start the name of your new XML physical format with 'XML', because this clearly identifies the type of the physical format that you are adding in relation to any of the other types.
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** or by pressing Ctrl+S.

Configuring XML Wire Format properties: Message sets

Configure the XML Wire Format properties of a message set using the Message Set editor.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Opening an existing message set" on page 83
- "Adding an XML wire format" on page 89

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 **XML Wire Formats** node of the Properties Hierarchy shows the name of each of the XML physical formats that have been added to the message set. If the physical format names are not in view, expand **XML Wire Formats** by clicking **+**.
3. Click the chosen XML physical format so that the properties of this format appear in the Details view of the Message Set editor.
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** or by pressing Ctrl+S. Alternatively click **File> Save All** or press Ctrl+Shift+S.

Renaming a physical format

Rename a physical format using the Message Set editor.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Opening an existing message set" on page 83

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 87 or "Adding an XML wire format" on page 89 or "Adding a TDS physical format" on page 88.

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 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 **XML Wire Formats**, **Custom Wire Formats**, or **Tagged/Delimited String Formats** by clicking **+**.
4. 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

Apply the default settings to a physical format layer that has previously been added to a message set.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Opening an existing message set" on page 83

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 87 or "Adding an XML wire format" on page 89 or "Adding a TDS physical format" on page 88.

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

You can remove a physical format layer from your message set.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83

- "Opening an existing message set" on page 83

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 87 or "Adding an XML wire format" on page 89 or "Adding a TDS physical format" on page 88.

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 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 **XML Wire Formats**, **Custom Wire Formats**, or **Tagged/Delimited Wire Formats**, by clicking **+**.
4. 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. Pressing Esc also returns you to the Broker Application Development perspective without making any changes.

Observing 2007 U.S. changes to daylight saving time

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Opening an existing message set" on page 83

This task assumes that you have added and configured one or more physical formats to existing message sets. For further information see: "Working with physical formats" on page 87.

This task describes how to ensure that the message sets observe daylight saving time (DST) in line with the 2007 U.S. changes.

If your message sets use a named time zone that is *not* changing DST in line with the 2007 U.S. changes, you do not need to do anything.

If you are using a GMT-04:00, GMT-05:00, GMT-06:00, GMT-07:00, or GMT-08:00 named time zone with DST, that must observe DST in line with the 2007 U.S. changes, do the following steps on every computer on which the broker is running:

1. Set the environment variable MQSI_USE_NEW_US_DST to an initial value: Y, for example.
2. Restart the broker to use the changed DST.

Configuring documentation properties: Message sets

Document a message set in the workbench.

Before you start:

Complete the following tasks:

- "Creating a message set" on page 83
- "Opening an existing message set" on page 83

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 **Message set**. The documentation text field appears in the Details view, with all the other logical properties of the message set.
3. Configure the documentation properties shown in the Details view to your requirements.

   You can use the Documentation property to set user-defined keywords and their value. These keywords are propagated to the Configuration Manager when you deploy the message set in a BAR file to a broker. These keywords are used to give additional information about the message set when you display deployed message set properties in the workbench. See "Message set version and keywords" on page 10 for more information.
4. Save your changes by clicking **File** → **Save**, or by pressing Ctrl+S. Alternatively, click **File** → **Save All**, or press Ctrl+Shift+S.

Deleting a message set

If you want to delete a message set from your message model, you must delete the message set project that contains the message set.

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the message set project folder that contains the message set that you want to delete and click **Delete** on the pop-up menu. This opens the Confirm Project Delete window, which asks whether you want to delete the message set project that you have specified.
3. Click **Also delete contents .....** to delete the contents of the message set project, or click **Do not delete contents** to cancel the deletion of the message set project. Pressing the Esc key on your keyboard also cancels the deletion of the message set project.

Important: When you delete a message set project, the action cannot be undone after you have confirmed the deletion. All folders and associated files for the message set project are deleted.

Applying a Quick Fix to a task list error

During the creation, migration and manipulation of message definition files, warnings or errors might occur; these are listed in the Problems view of the Broker Application Development perspective. Some of these warnings or errors can be cleared by applying a Quick Fix.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83

- "Creating a message definition file" on page 95

The types of warnings or errors that can be cleared using a Quick Fix are those where the construct has a broken link, where the construct has a facet that is not legal, 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 allows 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, then click **Quick Fix**. Note that **Quick Fix** might not be available for the problem that you are trying to fix.
4. Step through the windows that are displayed, making the selections that are required 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 is removed from the Problems view.

Working with a message definition file

Create, open, and delete a message definition file.

Before you start:

You must have completed the following task:

- "Creating a message set" on page 83

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" on page 95
- "Deleting a message definition file" on page 96

Opening an existing message definition file

This task topic describes how to open an existing message definition file in the Message Definition editor; you can then view and edit the contents of the file.

To open an existing message definition file:

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the message definition file (file extension `*.mxsd`) that you want to open, and select **Open**. This opens 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. However, you are advised to use only the workbench Message Definition editor to work with message definition files, because this editor correctly validates any changes that are 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

Creating an empty message definition file to contain your message model objects.

Before you start:

You must have completed the following task:

- "Creating a message set" on page 83

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.

You can create the message definition file in one of the following ways:

- Create the message definition file from scratch, see "Creating a message definition file from scratch."
- Base your message definition file on an existing resource (for example, an XML Schema file, an IBM® supplied message, an XML DTD file, a C header file, a COBOL file, or a WSDL file), see "Creating a message definition file from an existing resource" on page 96.
- Copy a message definition file from one message set to another.

  If you do copy a message definition file from one message set to another, you must check that the source and target message sets have identical physical formats, and that namespaces are enabled.

  If the source and target namespaces do not have identical formats, the physical format of the message definition file might be replaced by the default information applied to the target message set.

Creating a message definition file from scratch

Create an empty message definition file to contain message model objects.

Before you start:

You must have completed the following task:

- "Creating a message set" on page 83

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 from scratch:

1. Switch to the Broker Application Development perspective.
2. Open the New Message Definition File wizard.

   To do this, right-click on the message set project in the Broker Development view that you are adding the message definition file to, and click **New> Message Definition File** on the pop-up menu. The **Message Definition File** panel of the wizard is displayed. The target message set list is filtered to only show artifacts in the active working set.
3. Click on the message set, type a name into the File name field, and 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; you can use the editor to create your own message definitions. If you have chosen to use a target namespace, a directory structure that is 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 Broker Development view.

Creating a message definition file from an existing resource

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.

You must have completed the following task:

- "Creating a message set" on page 83

To create a new message definition file that is based on an existing resource:

1. Switch to the Broker Application Development perspective.
2. Open the appropriate New Message Definition File From wizard.

   Right-click on the message set project in the Broker Development view that you are adding the message definition file to, and click **New> Message Definition File From**. A submenu shows the list of resources from which you can choose.
3. Choose the resource on which to base your new message definition. Click **XML Schema File**, **IBM Supplied Message**, **XML DTD File**, **C Header File**, **COBOL File**, or **WSDL File**. The first panel of the corresponding wizard is displayed.
4. Step through the remainder of the wizard supplying the details as required.

The new message definition file, with the name that you have specified and a file extension of `*.mxsd`, opens in the Message Definition editor; you can use the editor to create your own message definitions. If you have chosen to use a target namespace, a directory structure that is 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 Broker Development view.

Deleting a message definition file

You can delete a message definition file from your message model.

To delete a message definition file from your message model:

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the message definition file (file extension `*.mxsd`) that you want to delete, then click **Delete**. Alternatively, select the message definition file that you want to delete in the Broker Development view, 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. Click **No**, or press the Esc key, to cancel the deletion of the message definition file.

Important: All files and objects that are associated with the message definition file are deleted. This action cannot be undone.

Working with message model objects

Add, configure, and delete objects.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95

This topic area describes the tasks that are involved in working with message model objects:

- "Adding message model objects"
- "Configuring message model objects" on page 108
- "Deleting objects" on page 122

Adding message model objects

Various tasks are involved in adding message model objects to a message definition file.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95

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 94 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" on page 98
- "Adding a message from a global element" on page 98
- "Adding a global element" on page 99
- "Adding a local element" on page 100
- "Adding an element reference" on page 100
- "Adding a wildcard element" on page 101
- "Adding a global attribute" on page 101
- "Adding a local attribute" on page 102
- "Adding an attribute reference" on page 103
- "Adding a wildcard attribute" on page 103
- "Adding a simple type" on page 104
- "Adding a complex type" on page 105
- "Adding a simple type to an element or attribute" on page 112
- "Adding a complex type to an element" on page 113
- "Adding a global group" on page 106
- "Adding an attribute group" on page 107

- "Adding a group reference" on page 107

Adding a message

Add a message to a message model.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

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 about configuring message model objects see "Configuring message model objects" on page 108.

Adding a message from a global element

Add a message from a global element to a message model.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94
- "Adding a global element" on page 99 (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 **Select a global element of complex type that is not already used for a message** 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 108.

Adding a message from a global type

Add a message from a global type to your message model.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94
- "Adding a global element" (This must be a global element of complex type)

To add a message from a global 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 **Messages** then click **Add Message From Global Type** on the pop-up menu to open the Add Message From Global Type window. This window lists all the global complex types that are not already associated with a message.
4. In the **Select a global complex type** list, click the global complex type 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 complex type 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 108.

Adding a global element

Add a global element to a message model.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

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 108.

Adding a local element

Add a local element to a message, type, group, or complex element.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

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, complex 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 about configuring message model objects see "Configuring message model objects" on page 108.

Adding an element reference

Add an element reference to a message, type, group, or complex element.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

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, complex 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 might 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 about Imports and Includes, see "Linking from one message definition file to another" on page 124.

You can now configure the element reference to your exact requirements. For further information about configuring message model objects see "Configuring message model objects" on page 108.

Adding a wildcard element

Add a wildcard element to a message, type, group, or complex element in a message model.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

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, complex 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 108.

Adding a global attribute

Add a global attribute to your message model.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

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 108.

Adding a local attribute

Add a local attribute to a message, complex type, or complex element.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

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, complex element, or attribute group) 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 about configuring message model objects see "Configuring message model objects" on page 108.

Adding an attribute reference

Add an attribute reference to a message, complex type, or complex element.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

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, complex element, or attribute group) to which you are adding the attribute reference then click **Add Attribute Reference** on the pop-up menu. This action 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 about configuring message model objects see "Configuring message model objects" on page 108.

Adding a wildcard attribute

Add a wildcard attribute to a message, complex type, or complex element.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

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, complex element, or attribute group) 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 about configuring message model objects see “Configuring message model objects” on page 108.

Adding a simple type

Add a simple type to your message model.

Before you start:

You must have completed the following tasks:

- “Creating a message set” on page 83
- “Creating a message definition file” on page 95
- “Opening an existing message definition file” on page 94

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 114.
- 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 option 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 option 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. It 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 option 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 option 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 option 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 option 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' referred to earlier.
  - **Add Local Member Type List**. This option 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' referred to earlier.
  - **Add Local Member Type Union**. This option 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 required position in the union. 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 about configuring message model objects see “Configuring message model objects” on page 108.

Adding a complex type

Add a complex type to your message model.

Before you start:

You must already have completed the following tasks:

- “Creating a message set” on page 83
- “Creating a message definition file” on page 95
- “Opening an existing message definition file” on page 94

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 108.

Adding a global group

Add a global group to your message model.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

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 about configuring the properties of message model objects see "Configuring message model objects" on page 108.

Adding a local group

Add a local group to a message, type, global group, or complex element.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

You can add a local group to a message, complex type, group, or complex element. This task assumes that you have previously added the relevant message, complex type, group, or complex element to your message model.

To add a local group to a message, complex 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, complex type, 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 about configuring message model objects, see "Configuring message model objects" on page 108.

Adding an attribute group

Add an attribute group to your message model.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

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 about configuring the properties of message model objects see "Configuring message model objects" on page 108.

Adding a group reference

You can add a group reference to a message, type, global group, or complex element.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

You can add a group reference to a message, complex type, group, or complex element. This task assumes that you have previously added the relevant message, complex type, group, or complex element to your message model.

To add a group reference to a message, complex 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, complex type, 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."

Adding an attribute group

Add an attribute group reference to a message model.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

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. An attribute group 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 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

Various tasks are involved in configuring message model objects

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Adding message model objects" on page 97 (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 94 for further details.

This topic area describes the tasks that are involved in configuring message model objects:

Renaming objects

You can rename message model objects in the workbench.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94
- "Adding message model objects" on page 97 (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 wildcards, local groups, or local types, because they do not have a name.

If an object can be renamed the usual way 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

Reorder message model objects within a message definition file.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94
- "Adding message model objects" on page 97 (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 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

You can copy an object in a message definition file, such as a message for a local element object, or types for a complex type object.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94
- "Adding message model objects" on page 97 (You must have added one or more objects to your message model)

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**. Alternatively, right-click the object in the Message Definition editor **Overview** tab, then click **Copy**.

The object is now copied.

Pasting objects

Paste objects that you have previously copied within the same message definition file

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94
- "Adding message model objects" on page 97 (You must have added one or more objects to your message model)
- "Copying objects" on page 110

You can paste objects that you have previously copied within the same message definition file.

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, you might be trying to paste to a location that is not valid. For example, you cannot paste a complex type into a local element.

Changing the type of an element or attribute

You can change the type to a local element, global element, local attribute, or global attribute.

Before you start:

You must already completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94
- "Adding message model objects" on page 97 (You must have added one or more objects to your message model)

You can change the type of an element or attribute 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 or attribute 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 lower-left corner of the editor area.
5. In the Properties Hierarchy click **Logical Properties → Global Element** (or **Logical Properties → Local Element**, **Logical Properties → Global Attribute**, or **Logical Properties → Local Attribute**). If necessary, expand **Logical Properties** by clicking **+**.
6. In the Details view, in the **Type** property, click the new type that 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 that you require, click **OK**.

The change to the type is applied wherever the element or attribute 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 or attribute" or "Adding a complex type to an element" on page 113.

Adding a simple type to an element or attribute:

You can add a simple type to a local element, global element, local attribute, or global attribute.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

This task assumes that you have previously added the relevant element or attribute to your message model.

To add a new simple 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 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** (or **Logical Properties > Local ElementLogical Properties > Global Attribute**, or **Logical Properties > Local Attribute**).
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:

You can add a complex type to a local element, global element, local attribute, or global attribute.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

This task assumes that you have previously added the relevant element or attribute to your message model.

When you add a complex type to an element or attribute, 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:

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** (or **Logical Properties > Local ElementLogical Properties > Global Attribute**, or **Logical Properties > Local Attribute**).
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**, 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.

Setting value constraints

You can set the value constraints associated with a simple type.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94
- "Adding a simple type" on page 104 or "Adding a simple type to an element or attribute" on page 112 (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 lower-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 XML Schema regular expressions.

See “Regular expression syntax” on page 778 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

You can configure the logical properties of an object that has previously been added to the message model.

Before you start:

You must have completed the following tasks:

- “Creating a message set” on page 83
- “Creating a message definition file” on page 95
- “Opening an existing message definition file” on page 94
- “Adding message model objects” on page 97 (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 lower-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.

   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 **FileSave All** or press Ctrl+Shift+S.

Configuring documentation properties: Message model objects

You can configure the documentation properties of an object that has previously been added to the message model.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94
- "Adding message model objects" on page 97 (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 lower-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

Working with the physical properties of message model objects.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94
- "Adding message model objects" on page 97 (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 87 or "Adding an XML wire format" on page 89 or "Adding a TDS physical format" on page 88.

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 120
- "Configuring TDS properties: Message model objects" on page 119
- "Applying default physical format settings: Message model objects" on page 121

Configuring Custom Wire Format (CWF) properties: Message model objects:

You can configure the Custom Wire Format (CWF) properties of a message model object by using the Message Definition editor

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94
- "Adding a Custom Wire Format (CWF)" on page 87
- "Adding message model objects" on page 97 (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 lower-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 Binary1 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 **FileSave All** or press Ctrl+Shift+S.

Configuring TDS properties: Message model objects:

You can configure the Tagged/Delimited String (TDS) properties of a message model object.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94
- "Adding a TDS physical format" on page 88
- "Adding message model objects" on page 97 (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 lower-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 Text1 but could have a user defined name instead.

5. Select the **Properties** tab located in the lower-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.

Configuring XML Wire Format properties: Message model objects:

You can configure the XML Wire Format properties of a message model object.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94
- "Adding an XML wire format" on page 89
- "Adding message model objects" on page 97 (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 lower-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.

Applying default physical format settings: Message model objects:

You can apply the default physical format settings to a message model object that is contained in a message definition file.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94
- "Adding message model objects" on page 97 (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 87 or "Adding an XML wire format" on page 89 or "Adding a TDS physical format" on page 88.

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 lower-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

Delete an object from your message model.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94
- "Adding message model objects" on page 97 (You must have added one or more objects to your message model)

To *remove* objects contained within a message definition file:

1. 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**.
2. In the Outline view, right-click the object that you want to remove then click **Delete** on the pop-up menu. Alternatively right-click the object in the Message Definition editor **Overview** tab, and then click **Delete**.

   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.
3. 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

A multipart message occurs when you embed a message in another message.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

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 105)
   - A global group (for further information on completing this task see "Adding a global group" on page 106)
   - A local group (for further information on completing this task see "Adding a local group" on page 106)

   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 113.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-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, you must click **Open**. For further information about using these three options, see "MRM content validation" on page 194.

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

Add an 'include', or an 'import' to the file that you want to reference.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Creating a message definition file" on page 95
- "Opening an existing message definition file" on page 94

There are two ways to link one message definition file to another: either you can add an 'include', or you can add an 'import', for the file that you want to reference.

To check whether a message definition file currently includes or imports other files:

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

Use the include option if you want to link to a message definition file with the same namespace, or if you want to link to a message definition file with no target namespace from a message definition file with a target namespace (chameleon behavior). You must also 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 lower-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 restriction 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 lower-left corner of the editor area.
5. In the Properties Hierarchy, right-click **Imports** then click **Add Import**. 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

Create a message category file to add categories that you can use to group different message sets.

Before you start:

Complete the following task:

To create a message category file:

1. Switch to the Broker Application Development perspective.

2. Right-click in the Broker Development view, then select **New → Message Category File** to open the New Message Category File wizard.

   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 you open the wizard.
3. In the first pane, select the Category Kind for the type of category that you are creating.
   - other. This value indicates that this message category represents a generic grouping of messages. The Category Usage field is disabled.
   - wsdl. This value indicates that this message category represents a WSDL operation. The specified category name is used as the WSDL operation name.

     Note: This use of categories is for compatibility with WebSphere Message Broker Version 6.0 only.
4. If you set Category Kind 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. The message set folder view is filtered to show only resources in the active working set.
6. In the **File name** field, type a name for the new message category file. The file is automatically given the file extension of `.category`.
7. Click **Next**. Select all 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 clear individual messages. You cannot complete the creation of the category file without adding one or more messages, and setting the **Role Type** and **Role Usage** values of each message correctly.
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.

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 125

To open an existing message category file:

1. Switch to the Broker Application Development perspective.

2. In the Broker Development view, 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

You can add a message to a message category file by using the Message Category editor.

Before you start:

You must have completed the following tasks:

- "Adding message model objects" on page 97 (to create at least one message)
- "Creating a message category file" on page 125
- "Opening an existing message category file" on page 126

To add a message to a message category file:

1. Switch to the Broker Application Development perspective.
2. Open the Message Category editor.
3. In the Properties Hierarchy, open the Add Messages window by right-clicking **Message Category**, then clicking **Add Messages**. The Add Messages window lists all the messages that are available for adding to the message category file. All messages that are in the message set but have not already been added to the category are displayed.
4. 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 clear individual messages.
5. 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 all additions that you make. To undo a change, right-click **Message Category** in the Properties Hierarchy, then click **Undo**. If you have added multiple messages, this action 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.
6. Save and validate the additions that you have made to the message category file by clicking **File → Save**, or by pressing Ctrl+S.

When you have saved the message category file after adding a message, you can no longer undo the addition of this message by using the **Undo** option. To remove a message after saving your changes, delete the message from the message category file.

When 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

Delete a message from a message category file.

Before you start:

To complete this task, you must have completed the following tasks:

- "Creating a message category file" on page 125
- "Opening an existing message category file" on page 126
- "Adding a message to a message category" on page 127

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 125
- "Opening an existing message category file" on page 126
- "Adding a message to a message category" on page 127 (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

You can delete a message category file from your message model.

Before you start:

To complete this task, you must have completed the following task:

- "Creating a message category file" on page 125

To delete a message category file:

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the message category file (`*.category` file extension) that you want to delete, then click **Delete**. Alternatively select the message category file in the Broker Development view, then either click **Edit → Delete**, or press the Delete key.
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: After you have deleted a message category file, the action cannot be undone.

Working with 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 131
- "Importing from COBOL copybooks" on page 133
- "Importing from IBM supplied messages" on page 135
- "Importing from WSDL" on page 136
- "Importing from XML DTD" on page 138
- "Importing from XML Schema" on page 140

The following tasks relate to importing using the command line:

- "Importing from the command line" on page 132 for C header files, COBOL Copybooks, XML DTDs and XML Schemas.
- "Importing WSDL definitions from the command line" on page 137

Importing file systems into the workbench

You can import file systems into the workbench by using the Import wizard, by dragging, or by copying.

Before the workbench can use files to create a message definition that is based on a WSDL definition, XML Schema, XML DTD, C header file, or COBOL copybook, you must import the files, or copy them into the local file structure.

Use one of the following options to import files for use by your selected message set project:

- "Using the Import wizard"
- "Dragging and dropping" on page 131
- "Copying and pasting" on page 131

You can then select the imported file in the New Message Definition File wizard to create a message definition that is based on the contents of this file.

Using the Import wizard

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 Broker Development view, click the project folder into which you are going to import the files.
3. Open the Import wizard by clicking **File** → **Import**.
4. On the Select page of the Import wizard, click either **File System** or **Archive 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. Either type the source name in the field, or click **Browse** and select the parent directory, or compressed file that contains the file or files that you want to import. Then click **OK** (directory) or **Open** (compressed file).

   Tip: Directories from which you have recently imported files, are shown in the list in the **Directory** field.
7. Using the left and right panes that appear under the **Directory** field, specify the folders or files, or both, that you want to import. Consider the following points when you are making your selections:
   - To import the entire contents of a folder, select the check box for this folder in the left pane. To view secondary folders within a folder, expand the folder by clicking the plus sign (+).
   - 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 grayed in the left pane to indicate that only some of the files in the folder will be imported.
   - To restrict the type of files that you are importing, click **Filter Types**, then, on the Select Types window, select the check boxes for the file types that 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 that are shown on the File System page, click **Select All**.
   - To clear all the folders and files that are currently selected on the File System page, click **Deselect All**.

   The **Select the destination for imported resources** field has already been set to the name of the project folder that you selected in step 2.

8. Optional: To change the destination project or folder, click **Browse** to open the Folder Selection window. Select an alternative project folder by clicking the folder, then clicking **OK**.
9. Optional: To overwrite existing resources and not have a warning displayed, select the **Overwrite existing resources without warning** check box. This check box applies to both compressed files and file systems.
10. File system import only: Select one of the following options, 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 shown in the Broker Development view under the project folder that you selected.

Dragging and dropping

You can use the drag-and-drop method to import files from your file system into the workbench. Drag the resources that you are importing to the exact location in the Broker Development view where you want the resources to be. Do not drag them onto a blank area in the Broker Development view.

To import files by dragging:

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 Broker Development view. When you are dragging resources into the Broker Development view, the project or folder into which you are trying to drop the resource is selected.
3. Ensure that the file or folder is copied into the workbench.

Copying and pasting

You can use the copy and paste function of your operating system 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 the copy and paste function in the operating system, copy the file or directory to your system clipboard.
3. Select the destination for the file or directory in the Broker Development view.
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 topic 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:

Complete the following tasks:

- "Creating a message set" on page 83

- "Importing file systems into the workbench" on page 129

Be aware of the following points:

- 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 can 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 you should add the physical layer *before* importing it to ensure that it is populated with settings from the C header file.
- You can import a C header file from the command line using mqsicreatemsgdefs.

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 this task, you should read the following information:

- 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 COBOL copybooks

This topic 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:

Complete the following tasks:

- "Creating a message set" on page 83
- "Importing file systems into the workbench" on page 129

Be aware of the following points:

- 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. You can then select the file in the New Message Definition File wizard.
- The wizard can 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 a Custom Wire Format (CWF) layer, or a Tagged/Delimited String (TDS) format 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 ensure that you add the physical layer *before* you import the data structure to ensure that it is populated with settings from the COBOL copybook.
- You can import a COBOL data structure from the command line using mqsicreatemsgdefs.

The steps below cover creating a 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 supplying 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 this task, you should read the following information:

- 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

You can create a new message definition file from an IBM supplied message.

Before you start:

You must have completed the following task:

- "Creating a message set" on page 83

The following steps describe how to create a new message definition file, and how to overwrite 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 From** on the workbench menu.
3. In the displayed list of options, select **IBM supplied message** and click **Next**.
4. Complete the fields of the panel that is displayed by the wizard. See "New message definition file wizard: IBM supplied message" on page 820.

When you have finished the import of the IBM supplied message:

- 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 that contains the message definition that you have created. 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 are displayed.

Importing from WSDL

You can use the New Message Definition File wizard in the workbench to create a new message definition from WSDL.

There are two methods for importing from WSDL:

- Create a message set and use the New Message Definition File wizard. This method is described here.
- Use the Start from WSDL and/or XSD files Quick Start wizard. See Creating an application based on WSDL or XSD files.

If you choose the first of these options, before you start you must have completed the following tasks:

- "Creating a message set" on page 83
- "Importing file systems into the workbench" on page 129

The following steps are required to create a completely new message definition file, or to overwrite 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 From ... wizard by clicking **File> New> Message Definition File From ...** on the workbench menu.
3. In the displayed list of options, select**WSDL file** and click **Next**. Alternatively, open the wizard by right-clicking a `.wsdl` file that was previously imported into the workbench and clicking **New> Message Definition File From ...** on the menu.
4. Step through the remainder of the wizard filling in the details as required.

   You must choose whether the WSDL file, or files, that you want to import are in the current workspace in the workbench or are outside the workspace.

   Check boxes provide options to:

   - Copy the source file (or files) into a directory of the message set project. By default, this check box is cleared.
   - Add the SOAP and XMLNSC domains to your message set so that you can use the SOAP nodes. By default, this check box is selected.

Note:

- The panels and options available in the wizard are dependant on the settings that you select.
- Some fields in the wizard might not be available. This might be because the field has a mandatory setting, or because the field has only one possible value, or because the field is not being used as a result of other settings that have been made.

When you have finished importing the WSDL file (or files) using the wizard:

- Check carefully 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 that contains the message definition that you have tried to create. The report has a `<wsdl-file-name>.wsdl.report.txt` file descriptor, where `<wsdl-file-name>` is the name of the WSDL definition that you are importing.
- Review the messages that are 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 using the New Message Definition File wizard by selecting the new option **IBM supplied message**.

Importing WSDL definitions from the command line

WSDL definitions can be imported using the (**mqsicreatemsgdefsfromwsdl**) command.

Before you start:

Before you attempt this task, read the following information:

- 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. It also allows you to add message definition files to an existing message set that is namespace enabled.

If you are adding new message definition files to an existing message set, the message set must have an XML physical format layer. To improve Web services interoperability, avoid unnecessary customization of the XML physical format layer for messages that participate in Web services processes.

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:

- 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.
- 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.

Before starting the import, the **mqsicreatemsgdefsfromwsdl** command copies the WSDL files that it needs into the workspace. These are the top level WSDL file and any further files that might be imported by it. The files are copied under the

specified message set in a folder called importFiles and are not removed after the import finishes. This allows you to update them, or run validation on them, in the workbench at a later time.

To import WSDL definitions using the command line:

1. Close the workbench. The workbench must not be running when you use the command line importer.
2. Invoke the **mqsicreatemsgdefsfromwsdl** command from a command prompt; you must specify the message set project name, the path name of the directory where the top level WSDL file is located, the name of that file, the location of the workspace, 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 the log file. The name of the log file is the name that you specified in the command, and it has the file extension `*.wsdl.report.txt`. This report is created when you invoke the **mqsicreatemsgdefsfromwsdl** command and, by default, it 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 by the **mqsicreatemsgdefsfromwsdl** command is visible in the project that you specified.

If an error occurs during the import of a WSDL definition, carefully check any errors that are reported. By default, all errors are written both 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

You can create a new message definition from an XML DTD by using the New Message Definition File wizard in the workbench.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Importing file systems into the workbench" on page 129

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 you should add the physical layer *before* importing it to ensure that it is populated with settings from the XML DTD.
- It is also possible to import an XML DTD from the command line using mqsicreatemsgdefs.
- The file extension must be `.dtd` in lowercase.

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 this task, you should read the following information:

- 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

You can use the New Message Definition File wizard in the workbench to create a new message definition from an XML Schema

Before you start you must have completed the following tasks:

- "Creating a message set" on page 83
- "Importing file systems into the workbench" on page 129

Before you begin this task, you should be aware of the following points:

- To create a new message definition file from an XML Schema, the schema file must already exist 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 you are importing a collection of related XML Schema files, you are advised to use the mqsicreatemsgdefs command. This imports all the XML Schema files in a single operation, and automatically adjusts the import and include paths.
- 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. For this reason, you should 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. For this reason, you should add the physical layer to the message set before importing the schema. This ensures that the message set is populated with the settings and values from the schema. The XML Schema is not modified to remove namespaces.
- 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 lowercase.

The following steps create a completely new message definition file or overwrite 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 that was 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, and then click **Next**.
4. Step through the remainder of the wizard, filling in the details as required. The processing time for importing the XML Schema varies according to the size and complexity of that schema. In a large and complex schema, it can take some time to import the file, generate the log file and display any task list warnings or errors.

When you have finished importing the XML Schema using the wizard:

- Carefully check the log file for any warnings or errors in the report that is created when the file is imported. These warnings and error messages give information about whether the schema failed to import or needed to be modified to enable it to be successfully imported. You can find this report in the `log` directory structure within the project that contains the message definition that you have tried 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 that are shown in the workbench task list to check whether any new warnings or error messages have appeared. Although you might have

imported a perfectly valid schema, the task list will display warnings or error messages for any errors that exist in the message definition file. 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 122. Deleting the constructs also removes the warning messages from the task list. If you decide not to delete the constructs, they remain in the message model but are not be deployed to the broker, or used for any other purpose. The warning messages remain in the task list, but you can use the message model normally.
- If the XML Schema that you are importing contains xsd:redefine constructs, error messages appear in the 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 the XML Schema that you are importing contains xsd:attribute constructs that contain both a fixed value and a default value, error messages appear in the task list to tell you that this construct is unsupported. However, the schema is still imported and the fixed value is used, not the default value. The error messages can be ignored.
- If you are importing a collection of related XML Schema files and the Message Definition Editor cannot 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 more 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 this task, you should read the following information:

- 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 documentation from message sets and message flows

You can generate documentation from your message sets, message flows, message definition files, message maps, Java™ files, ESQL files, and deployable WSDL files.

To generate documentation that describes your message sets, message flows, message definition files, message maps, Java files, ESQL files, and deployable WSDL files:

1. Switch to the Broker Application Development perspective.
2. In the context menu of the Broker Development view, right-click a message set project, a message set, a message flow, a message definition file, a Java file, an ESQL file, or a deployable WSDL file, and select the action **Generate Documentation**. The Documentation Generation wizard opens.
3. Provide the information that is requested to describe the documentation report that you want, and click **Next** to move to the next panel of the wizard.
4. Step through the wizard, clicking **Next** to move to a new panel, and clicking **Finish** when you have described all the information that you want your report to document.

Generating XML Schemas

You can generate either a single XML Schema from a message definition file, or multiple XML Schemas from a message set.

To generate a single XML Schema from a message definition file, see “Generating an XML Schema” on page 145.

To generate multiple XML Schemas (one from each message definition file in a message set) see “Generating XML Schemas.”

Generating XML Schemas

You can generate an XML Schema for each message definition file in a message set.

You must have completed the following tasks:

- “Creating a message set” on page 83
- “Working with a message definition file” on page 94
- “Working with message model objects” on page 97

Note: WebSphere Message Broker uses XML Schema 1.0 to describe the logical structure of messages.

Tip: You should replace any deprecated constructs before you generate XML Schema representations of your models.

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the message set folder from which you want to generate XML Schemas, and click **Generate> XML Schemas**.
3. The Generate XML Schemas window is displayed, and you must put into the **Zip file name** field the name of the compressed file (`*.zip` file extension) that you want to contain the generated XML Schemas.
4. Select a destination folder for this compressed file. You can choose a location either inside or outside the workspace:
   - Click **Create in a workspace directory** and select the required destination folder from the expanded workspace directory. The contents of the folder that you select are overwritten.

     If you want to create a new folder:

     a. Click on the desired location.

     b. Click **Create New Folder**.

     c. Click OK
   - Click **Export to an external directory** and click **Browse** to expand the directory. Select a folder from the expanded directory. The contents of the folder that you select are overwritten.

     If you want to create a new folder:

     a. Click on the desired location.

     b. Click **Make New Folder** and type the name of the new folder into the directory tree.

     c. Click OK
5. Optional: Choose from the list given in the **XML Wire Format** field an XML wire format that you want to use to generate the XML Schemas.

   Tip: You must have previously added one or more XML Wire Format layers to the message set if you want to use an XML physical format when you generate XML Schemas. For further information see “Adding an XML wire format” on page 89.

6. If you do not want strict generation of an XML Schema, clear the **Strict generation** check box at the bottom of the Generate XML Schemas page. By default, this check box is selected.

   Tip: For further information on strict and lax generation of an XML Schema, see "Generate XML schema" on page 79.
7. Click **Finish**. The compressed file that contains your generated XML Schemas is created.

Generating an XML Schema

You can generate an XML schema from a message definition file.

Before you start:

You must have completed the following tasks:

- "Creating a message set" on page 83
- "Working with a message definition file" on page 94
- "Working with message model objects" on page 97

Note: WebSphere Message Broker uses XML Schema 1.0 to describe the logical structure of messages.

Tip: You should replace any deprecated constructs before generating XML Schema representations of your models.

This task topic describes how to generate an XML Schema from a message definition file:

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the message definition file (`*.msxd` file extension) from which you want to generate an XML Schema, then click **Generate → XML Schema** on the menu.
3. The Generate XML Schema window is displayed, and the message definition file that you selected is highlighted. The message definition file list is filtered to only show artifacts in the active working set. If this is not the message definition file from which you want to generate an XML Schema, select the correct message definition file.
4. Optional: From the drop down list at the bottom of the Generate XML Schema window, select the XML Wire Format that you want to use to generate the 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 89.
5. If you do not want strict generation of an XML Schema, clear the **Strict generation** check box at the bottom of the Generate 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 79.
6. Click **Next** to move to the next page of the wizard.
7. Select a destination folder for the XML Schema. You can choose a location either inside or outside the workspace:

- Click **Create in a workspace directory** and select the required destination folder from the expanded workspace directory. The contents of the folder that you select are overwritten.

  If you want to create a new folder:

  a. Click on the desired location.

  b. Click **Create New Folder**.

  c. Click OK
- Click **Export to an external directory** and click **Browse** to expand the directory. Select a folder from the expanded directory. The contents of the folder that you select are overwritten.

  If you want to create a new folder:

  a. Click on the desired location.

  b. Click **Make New Folder** and type the name of the new folder into the directory tree.

  c. Click OK

8. Click **Finish**. Your XML Schema is generated.
9. Use the Broker Development view 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 menu. This opens the schema editor.

   Tip: The **Design**, **Source** or **Graph** tabs located in lower-left corner of the schema editor provide you with different views of generated XML Schema.

Generating a WSDL definition from a message set

To ensure the highest interoperability of your Web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Before you start you must already have completed the following tasks:

- "Creating a message set" on page 83

Replace any deprecated constructs before generating WSDL representations of your message models.

To generate a WSDL definition:

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the folder that contains the message set file from which you want to generate a Web service definition, and select **Generate → WSDL Definition**. This starts the Generate WSDL wizard.
3. Step through the wizard filling in the details as required. Some of the panels and options are subject to settings that you make within the wizard and might not always be shown. Also, some fields in the wizard might be greyed out. This happens when a field has a mandatory setting, or when the field is not used because of settings that have already been made in other fields.

   By default, the wizard creates the WSDL in the message set project. If you are going to use the WSDL to configure a SOAP node, create the WSDL in the message set, not the message set project.

On completion of the Generate WSDL wizard, you have generated a WSDL definition. The file extension for WSDL files is .wsdl, and the file extension 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 WSDL that is generated for a JMS binding:

```
<wsdl:service name='HTTP'>
  <wsdl:port binding='tns:JMSSoapBinding' name='HTTP'>
    <wsdlsoap:address
       location='jms:/queue?destination=jms/MyQueue&amp;
                 connectionFactory=jms/MyCF&amp;
                 priority=5&amp;
                 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

Message model reference information

Reference information in this section can help you develop and configure message models.

Message model reference information is available for:

- "Message set preferences"
- "Message set properties" on page 153
- "Message definition file properties" on page 185
- "Message category properties" on page 187
- "Message model object properties" on page 188
- "Deprecated message model object properties" on page 615
- "Additional MRM domain information" on page 746
- "Additional MIME domain information" on page 791
- "Generated model representations" on page 801
- "Import formats" on page 806
- "Message model wizards" on page 818

Message set preferences

Preferences for message sets.

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:

- "Message Set Editor and Message Definition Editor preferences"
- "Validation of the message model" on page 152
- "XML Schema Importer" on page 153

Message Set Editor and Message Definition Editor preferences

While looking at a large message set that contains a number of message definition files that have different namespaces, or multiple message definition files that have the same namespace, you might want to view the information in alternative ways to make it easier for you to visualize the structure of the message set. If you double click on the global construct, you open the message definition file in which the global construct is defined.

Message set editor settings

Property	Type	Meaning
Group by namespace and then by collections	Button	Selecting this view groups the global constructs by namespace 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) 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.
Prefix for created messages	String	This property allows you to specify a prefix to precede the name of the initial complex type in the name of the created message. This prefix applies only to messages created from C or COBOL files. The default value is `msg_`. Note, however, that no prefix is applied when a message is created from a C file, and the selected preprocessing option is SAP ALE IDoc or SAP File IDoc.

Tab Extensions

Click **Tab Extensions** to display check boxes that allow you to determine what tabs are enabled for the Message Set Editor, the Message Definition Editor, and the Message Category Editor. All these check boxes are always selected and cannot be cleared.

Editor	Tab Extensions
Message Set Editor	Properties
Message Definition Editor	Overview Properties
Message Category Editor	Properties

A control is provided that allows you to choose the order in which the tab extensions for each of the editors are displayed.

Validation of the message model

You can customize some of the warning messages that are generated by message set validation. Use the Message Set Validation Preference page to do this.

Any warning or error that falls into any of the categories that are listed below can be customized according to the relevant category. The customization can affect both severity and priority.

The severity can be one of the following values:

- Error
- Warning
- Info
- Ignore

If the severity is not *Ignore*, the priority can be one of the following values:

- High
- Normal
- Low

If the severity is *Ignore*, you cannot change the priority.

Message set validation settings

The following is a list of the categories that you can customize:

- 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
- Unique Particle Attribution checks
- 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

Preferences for the message set 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

Message sets have properties that you can set to define their characteristics and the way in which they are processed.

General message set properties

The table below defines the properties that you can set to customize the message set.

Property	Type	Meaning
Default message domain and Supported message domains	String and check boxes	The message parser name must match the *Message Domain* property of any input node that processes messages from the message set, or the <Msd> element value of any MQRFH2 header that precedes a message from the message set. Choose a value from the list offered for the *Default Message Domain* property, and select check boxes (from *Supported Message Domains*) to choose other domains. You can select as many of these check boxes as you want. Use the message parser name when you write 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 they generate ESQL field references. You can choose from the following names: • XMLNSC (the default if you select **Finish** from page two of the New Message Set wizard). Choose this domain if you want to model XML messages. You can deploy the message set to brokers if you want, because the XMLNSC parser optionally uses the message set at run time. • MRM. Choose this domain for binary or text messages. You can also use this domain for XML messages. You must deploy the message set to the brokers that receive these messages. The deploy action creates a runtime dictionary against which the MRM parser checks the received message. • SOAP. Choose this domain for SOAP Web Services. • DataObject. Choose this domain for data from WebSphere Adapters. • XMLNS. You might need to choose this domain for some kinds of XML messages. You do not have to deploy the message set to brokers, because the XMLNS parser does not use the message set at run time. • JMSMap. Choose this domain if you want to model a JMS MapMessage message. You do not have to deploy the message set to brokers, because this parser does not use the message set at run time. • JMSStream. Choose this domain if you want to model a JMS StreamMessage message. You do not have to deploy the message set to brokers, because this parser does not use the message set at run time. • MIME. Choose this domain if you want to model a MIME message. You do not have to deploy the message set to brokers, because the MIME parser does not use the message set at run time. • XML. This domain is deprecated. Use the XMLNSC domain instead. • IDOC. This domain is deprecated. Use the MRM domain instead.

Property	Type	Meaning
Use namespaces	Check box	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. By default, this check box is selected. Using namespaces affects how elements are created in the 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 the Java that you write. Always select this property if you want to use the message set to model XML messages.

MRM domain

Property	Type	Meaning
Default wire format	String	(Optional) Specify the default wire format used, only if you select MRM as the default message domain, or MRM is selected in the list of supported message domains. The default value is <no default specified>. If you do not select MRM, either as the default message domain or as one of the supported message domains, the Default Wire Format property is unavailable.
Message set ID	String	This property 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. Using this technique, 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.
Message type prefix	String	This property is used when you define multipart messages, specifically when using the Message Path technique to identify embedded messages. 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 *Message Type* property that is specified for the outermost message (specified either in the MQRFH2 header of the message, or in the input node of the message flow). If you set a value, it must be in the form id1/id2/.../idnu 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). The table below, describing the use of the message set property *Message Type Prefix*, shows how this value is combined with the *Message Type* property of an input message.
Broker will treat Length facet as MaxLength	Check box	Select this property if you want the COBOL importer to create a maxLength facet, rather than a length facet, for a fixed length string element. By default, this check box is selected.

Use of the *Message type prefix* property

The table below shows the implications of using the property *Message type prefix*. The message type or message prefix can describe either 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.

If you are using MRM or IDOC domains, in addition to the main message set properties, you can update message set properties that are specific to each of the physical formats. Links to reference topics that describe these properties are given below.

Custom Wire Format message set properties

The tables define the properties that you can set for a Custom Wire Format 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 Boolean False Value. The default value is 00000001.

Property	Type	Meaning
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 Boolean True Value. 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 Boolean True Value or Boolean False Value, or different. The default value is 00000000.

Output settings

Use these settings when messages are being produced.

Property	Type	Meaning
Byte Alignment Pad	String	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: • Select SPACE, NUL, or 0 (the default) from the list of values shown. • Enter a character between quotation marks, for example "c" or 'c', where c is any alphanumeric character. • Enter a decimal character code in the form YY where YY is a decimal value. • Enter a hexadecimal character code in the form 0xYY where YY is a hexadecimal value. • Enter a Unicode value in the form U+xxxx where xxxx is a Unicode value specified in hexadecimal format.
Policy for Missing Elements	Enumerated	This property determines the action that is taken by the broker when fields are missing from the message tree when the message is serialized (for output): • Use *Default Value* (the default). If a *Default Value* exists for the element, write it; otherwise, throw an exception. • Use *Null Value*. If the *Nillable* property of the element is selected, and an *Encoding Null Value* is specified for the element, write the *Encoding Null Value* according to the rules that are defined by the *Encoding Null* property; otherwise, throw an exception.

Property	Type	Meaning
Truncate fixed length strings	Check box	This property applies only to output strings. If this check box is selected, and the element or attribute is a fixed length string (that is, the logical type is xsd:string and the physical type is Fixed Length String) that is longer than either the length that is specified in the model or the length reference, the string is truncated to this length. No exception is raised on output, unless validation (see Validating messages) is active. The end from which data is truncated is determined by the value of the *Justification* property. If the value of the *Justification* property is Left justify, data is truncated from the right; if the value of the *Justification* property is Right justify, data is truncated from the left. However, if the value of the *Justification* property is Not applicable, truncation does not occur and an exception occurs if the string is too long. If this check box is cleared, an exception occurs if the element or attribute is a fixed length string (that is, the logical type is xsd:string and the physical type is Fixed Length String) that is longer than either the length that is specified in the model, or the length reference. This behavior occurs in releases of the WebSphere Message Broker earlier than Version 6.1. By default, this check box is cleared.

Binary representation of decimal values

Property	Type	Meaning
Packed Decimal Positive Code	Enumerated	Select, from the list, the positive sign that is used for packed decimal numbers. The default value is C, which indicates that `0x0C` is used as the positive sign; this value is used in most systems. You can also select F, which indicates that `0x0F` is used as the positive sign; this value is used in some systems.

Datetime settings

Property	Type	Meaning
Derive default dateTime format from logical type	Button	Select this option if you want the default dateTime format to be determined by the logical type of the element or attribute. You can override this property for an element or attribute within a complex type.
Use default dateTime format	Button and String	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. You can override this property for an element or attribute within a complex type. For more information, see “DateTime formats” on page 782.
Start of century for 2-digit years	Integer	This property determines how 2-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 2-digit dates are interpreted as being in the range 1989 - 2088.

Property	Type	Meaning
Days in First Week of Year	Enumerated	Specify the number of days of the new year that must fall within the first week. The start of a year typically 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; therefore, week 1 starts some days into the new year. Otherwise, it is considered to be the first week of the new year; in this case, week 1 starts some days before the new year. Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a number from the list that is displayed.
First Day Of Week	Enumerated	Specify the day on which each new week starts. Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a value from the list that is displayed.

Property	Type	Meaning
Strict DateTime Checking	Check box	Select this option if you want to restrict dateTimes to a valid dateTime format. If Strict DateTime Checking is selected, receiving an incorrect dateTime causes an error. **Strict dateTime checking** Examples of strict dateTime checking are: • DateTimes are restricted to valid dateTimes only. When you use this option, a date such as the 35th March is not processed as 4th April, and 10:79 is not processed as 11:19. Receiving an out-of-band dateTime, such as these examples, causes an error to occur. • 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 that you require. The maximum number of digits that is permitted becomes the upper bound for a particular symbol. For example, day in month has an upper bound of 31; therefore, a format string of 'd' allows the values 2 and 21 to be parsed, but does not allow the values 32 and 210. On output, numbers are padded with zeros to the specified length. A year is a special case; see the message set property 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. • White space is not skipped over. The white space in the input string must correspond with the same number and position of white space in the formatting string. • If data remains that is not parsed in the input string after all the symbols in the formatting string have been matched, an error occurs. **Lenient dateTime checking** Examples of lenient dateTime checking are: • The parser converts out-of-band dateTime values to the appropriate in-band value. For example, a date of 2005-05-32 is converted to 2005-06-01. • Output of dateTimes always adheres 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. Therefore, dates that are not valid - for example, 2005-1-123 and 2005-011-12 - can be entered. The first value of 2005-1-123 is output as the date 2005-05-03, and the second value of 2005-011-12 is output as the date 2005-11-12. • The number of the timezone formatting symbol Z is applicable only to the output dateTime format. • White space is skipped over.
Time Zone	Enumerated	The value that you set for this property is used if the value that you specified for the Default DateTime Format property does not include Time Zone information. The initial value is Use Broker Locale, which causes the broker to get the information from the underlying platform. You can change this property by selecting from the list of values.
Daylight Saving Time	Check box	Select this option if the area in the Time Zone property observes daylight saving time. If it does not observe daylight saving time, do not select this option. For example, if an area is selected in Time Zone and this option is not selected, the value passed represents the time zone without the daylight saving time.

Property	Type	Meaning
Use input UTC format on output	Check box	This property applies to elements and attributes of logical type xsd:dateTime or xsd:time that contain a dateTime as a string and that have a dateTime format of I, IU, T, or TU, or that include ZZZ or ZZZU. Such elements and attributes can specify Coordinated 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. If this property is selected, and the element or attribute is copied to an output message, the UTC format is preserved into the output message and overrides the format that is implied by the dateTime format property. If this property is cleared, or if the element or attribute was not copied from an input message, the UTC format in the output message is controlled solely by the dateTime format property.

Character and numeric encoding for non-WebSphere MQ messages

Use these settings only for messages with no MQMD.

Property	Type	Meaning
Default CCSID*	Integer	Enter a numeric value for the default Coded Character Set Identifier. The default is 500. If the input message is a WebSphere MQ message, the equivalent attribute that is set for the queue manager is used, and this property is ignored.
Default Byte Order*	Enumerated	Select either Big Endian (the default) or Little Endian from the list to specify the byte order of numbers that are represented as binary integers. 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. Your choice must match the encoding with which messages are created. Typically, Big Endian is the correct option for messages that are created on UNIX® or z/OS®; Little Endian is the correct option for messages that are created on Windows®. Do not use this property 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 encoding of the broker queue manager.
Default Packed Decimal Byte Order*	Enumerated	Select Big Endian (the default) or Little Endian from the displayed 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. Your choice must match the encoding with which messages are created. Typically, Big Endian is the correct option for messages that are created on UNIX or z/OS; Little Endian is the correct option for messages that are created on Windows.
Default Float Format*	Enumerated	Select one of S390 (the default), IEEE, or Reverse IEEE from the displayed list to specify the byte order of numbers in the message that are represented as floating point.

TDS Format message set properties

The following tables show the properties that you can set for a TDS format message set.

See "Default TDS message set properties" on page 172 for the default settings of these properties for each of the industry standards.

Messaging Standard

Property	Type	Meaning
Messaging Standard	Enumerated	Specify the standard to be used for this wire format. Select one of the following values: • User Defined Text - for text data not based on a standard • SWIFT • ACORD AL3 • EDIFACT • X12 • TLOG • HL7 • CSV - Comma Separated Values • User Defined Mixed - for mixed text and binary data If you are defining your own tagged/delimited messages, or are using a standard that is not included in the list of values shown, select either User Defined Text, if all your data is text, or User Defined Mixed, if not all your data is text. The value that you select for this property determines the default values of some of the other properties. The default is User Defined Text.

Data element separation settings

Property	Type	Meaning
Group Indicator	String	Specify the default value of a special character or string that precedes the data that belongs to a group or complex type within the bit stream.
Group Terminator	String	Specify the default value of a special character or string that terminates data that belongs to a group or a complex type within the bit stream.
Delimiter	String	Specify the default value of a special character or string that specifies the delimiter that is used between data elements. This property applies only to the delimited Data Element Separation methods (Tagged Delimited, All Elements Delimited, and Variable Length Elements Delimited).

Property	Type	Meaning
Suppress Absent Element Delimiters	Enumerated	Use this property to select whether you want delimiters to be suppressed for elements that are missing within a message. Select from: • 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 three elements and only two are present, the last delimiter can be omitted from the message. • Never. Use this option to ensure that even if optional elements are not present, all delimiters are written out. This option must be used when the same delimiter is used to delimit parent objects and child objects. For example, if an optional child element is missing and all the delimiters are the same, message processing applications cannot tell where the child elements in a message ends and where the next parent element starts.
Tag Data Separator	String	Specify the default value of a special character or string that separates the tag from the data. If you set the property Tag Data Separator, the Length of Tag property is ignored. This property applies only to the tagged Data Element Separation methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).
Length of Tag	Integer	Specify the default length of a tag value. When the message is parsed, this property allows tags to be extracted from the bit stream. The Tag Data Separator and Length of Tag properties are mutually exclusive. If you set the property Tag Data Separator, the Length of Tag property is ignored. This property applies only to the tagged Data Element Separation methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).

Note: Any value that you set for a group or complex type property overrides the value that you set for the corresponding message set property.

Character data settings

Property	Type	Meaning
Default CCSID	Integer	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. 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).

Property	Type	Meaning
Trim on input	Enumerated	This property applies only to elements and attributes with a physical type of Text. This property specifies whether a simple element or attribute value is to be trimmed when it is parsed. The property does not apply to a simple element, or attribute, with a logical type of Boolean or Binary. All trimming is applied to element or attribute values before the conversion of the value to its logical type. This property does not apply when writing elements or attributes. This property only applies to a simple element, or attribute, that is contained within a complex type or group that has the Justification property set to Left Justify or Right Justify, and that satisfies one of the following conditions: • The Data Element Separation property is set to Fixed Length, Fixed Length AL3, Tagged Fixed Length, Use Data Pattern, or Tagged Encoded Length. • The Data Element Separation property is set to Variable Length Elements Delimited, and the element or attribute has a value set for its model length or length reference. • The Data Element Separation property is set to Tagged Delimited or All Elements Delimited, and the Observe Element Length property is set. The element or attribute has a model length or length reference value set. This property can be set to one of the following values: • No Trim. No characters are trimmed from the element or attribute value. • Leading White Spaces. White space characters are trimmed from the left of the element or attribute value. • Trailing White Spaces. White space characters are trimmed from the right of the element or attribute value. • Trim Both. White space characters are trimmed from both the left and the right of the element or attribute value. • Trim Padding Chars. Padding characters are trimmed from the element or attribute value. The padding character is set by the Padding Character property of the element or attribute. If the Justification property of the element or attribute is set to Left Justify, the padding characters are trimmed from the right. If the Justification property of the element or attribute is set to Right Justify, the padding characters are trimmed from the left. If the Justification property of the element or attribute is set to Not Applicable, no trimming takes place. White space characters include control characters that are in the range from U+0000 to U+001f and from U+007f to U+009f. You might need to use this property 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 when you process these fields.

Property	Type	Meaning
Truncate on output	Check box	This property applies only to output strings that have a physical type of Text. The property applies to elements or attributes that have a logical type of xsd:string and that are contained within a structure with a Data Element Separation of Fixed Length, Fixed Length AL3, Tagged Fixed Length, Use Data Pattern, or Variable Length Elements Delimited where a length has been specified. If this check box is selected, and the element or attribute has a length that is longer than the length that is specified in the model or the length reference, the string is truncated to this length. No exception is raised on output, unless validation (see Validating messages) is active. The end from which data is truncated is determined by the value of the Justification property. If the value of the Justification property is Left justify, data is truncated from the right; if the value of the Justification property is Right justify, data is truncated from the left. However, if the value of the Justification property is Not applicable, truncation does not occur and an exception occurs if the string is too long. If this check box is cleared, an exception occurs if the element or attribute is a fixed length string (that is, the physical type is Text and a length has been specified) that is longer than either the length that is specified in the model or the length reference. This behavior occurs in releases of the WebSphere Message Broker earlier than Version 6.1.
Escape Character	Button and String	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. Escape characters apply only in variable length fields. Escape characters, on parsing, always escape the next character, and are always removed. Escape characters, on writing, are inserted in front of all the characters that are listed in Reserved Characters. You can specify either an escape character or a quote character, but not both, for a given message set.
Quote Character	Button and String	Specify the quote 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. Quote characters apply only to variable length fields. Quote characters, on parsing, must be present at both the start and the end of the data, and are always removed. Quote characters, on writing, are added to both the start and end of the data, if the data contains any character that is listed in the Reserved Characters property. You can specify either an Escape Character or a Quote Character, but not both, for a given message set.

Property	Type	Meaning
Reserved Characters	String	Specify any special reserved characters. Either these reserve characters must be preceded by the Escape Character, or the data field that contains them must be delimited by a pair of Quote Characters, if they are to be included as part of the data. The Escape Character, Quote Character, delimiters, and group indicators must be included in this list. If the set of reserved characters is to be updated dynamically (in the case of EDIFACT and X12 when reserved characters, such as delimiters, are specified in service strings), you must use the supplied mnemonics to specify characters in this list. If you have specified Reserved Characters, an Escape Character or a Quote Character must also be specified. Reserved characters apply only in variable length fields. Reserved characters are not used when parsing.

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.
Packed decimal positive code	String	Controls the positive sign that is used for packed decimal fields. Valid values are C or F.Specify the character that is used to separate the whole part of a number from its fraction.
Strict Numeric Checking	Check box	Use this property in conjunction with the Messaging Standard property, the Virtual Decimal Point property and the Precision property of an element. Using this property allows you to apply stricter rules for the checking of numbers. The rules for Strict Numeric Checking are: • If the Precision property of an element is set to All Significant Digits , a decimal separator is present only if the value has a fractional part. • If the Precision property of an element is set to Explicit Decimal Point, the decimal separator must always be present, even if the fractional part is missing. • If the Precision property of an element is set to Exponential Notation, the incoming value must be in exponential notation. Exponential notation is only allowed for floating numbers. • If the Precision property of an element is set to a specific value, the specific number of digits after the decimal separator must be present. • All values must contain at least one digit in the integer part of the number. • If a Virtual Decimal Point of an element has been set, the number must not have a decimal point. • Except for EDIFACT, the decimal separator can be only the specified value, and '.' is not permitted. For EDIFACT, both '.' and the specified separator are permitted. In this case, the decimal separator must be specified as ',' and the code permits '.' to be used. • Except for exponential functions, only digits 0-9, the decimal separator, the positive sign, and the negative sign 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.

Property	Type	Meaning
Derive sign from logical type	Check box	If this property is selected, an unset TDS Signed property attempts to derive its value from the simple type of the element (integer and decimal simple logical types only). For these logical types it applies only to the Integer, External Decimal, and Packed Decimal physical types.
Default byte order	Enumerated	Controls the byte order of numbers that are represented as binary integers for messages with no MQMD. Valid values are Big Endian or Little Endian. This property stores the default byte order for numbers that are represented as binary integers for messages with no MQMD, but this value can be overridden when the message is processed.
Default packed decimal byte order	Enumerated	Controls the byte order of numbers that are represented as packed decimal for messages with no MQMD. Valid values are Big Endian or Little Endian. This property stores the default byte order of numbers that are represented as packed decimal for messages with no MQMD, but this value can be overridden when the message is processed.
Default float format	Enumerated	Controls the format of numbers that are represented as float for messages with no MQMD. Valid values are S390, IEEE, or Reverse IEEE. This property stores the default format of numbers that are represented as float for messages with no MQMD, but this value can be overridden when the message is processed.

Representation of boolean values

Property	Type	Meaning
Text boolean true value	String	Specifies the character that represents the text Boolean true value.
Text boolean false value	String	Specifies the character that represents the text Boolean false value.
Text boolean null value	String	Specifies the character that represents the text Boolean null value.
Binary boolean true value	String	Specifies a hexadecimal value that represents the binary Boolean true value.
Binary boolean false value	String	Specifies a hexadecimal value that represents the binary Boolean false value.
Binary boolean null value	String	Specifies a hexadecimal value that represents the binary Boolean null value.

Datetime settings

Property	Type	Meaning
Derive default dateTime format from logical type	Button	Select this option if you want the default dateTime format to be determined by the logical type of the element or attribute. You can override this property for an element or attribute within a complex type.

Property	Type	Meaning
Use default dateTime format	Button and String	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. You can override this property for an element or attribute within a complex type. For more information, see "DateTime formats" on page 782.
Start of century for 2-digit years	Integer	This property determines how 2-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 2-digit dates are interpreted as being in the range 1989 - 2088.
Days in First Week of Year	Enumerated	Specify the number of days of the new year that must fall within the first week. The start of a year typically 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; therefore, week 1 starts some days into the new year. Otherwise, it is considered to be the first week of the new year; in this case, week 1 starts some days before the new year. Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a number from the list that is displayed.
First Day Of Week	Enumerated	Specify the day on which each new week starts. Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a value from the list that is displayed.

Property	Type	Meaning
Strict DateTime Checking	Check box	Select this option if you want to restrict dateTimes to a valid dateTime format. If Strict DateTime Checking is selected, receiving an incorrect dateTime causes an error. **Strict dateTime checking** Examples of strict dateTime checking are: • DateTimes are restricted to valid dateTimes only. When you use this option, a date such as the 35th March is not processed as 4th April, and 10:79 is not processed as 11:19. Receiving an out-of-band dateTime, such as these examples, causes an error to occur. • 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 that you require. The maximum number of digits that is permitted becomes the upper bound for a particular symbol. For example, day in month has an upper bound of 31; therefore, a format string of 'd' allows the values 2 and 21 to be parsed, but does not allow the values 32 and 210. On output, numbers are padded with zeros to the specified length. A year is a special case; see the message set property 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. • White space is not skipped over. The white space in the input string must correspond with the same number and position of white space in the formatting string. • If data remains that is not parsed in the input string after all the symbols in the formatting string have been matched, an error occurs. **Lenient dateTime checking** Examples of lenient dateTime checking are: • The parser converts out-of-band dateTime values to the appropriate in-band value. For example, a date of 2005-05-32 is converted to 2005-06-01. • Output of dateTimes always adheres 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. Therefore, dates that are not valid - for example, 2005-1-123 and 2005-011-12 - can be entered. The first value of 2005-1-123 is output as the date 2005-05-03, and the second value of 2005-011-12 is output as the date 2005-11-12. • The number of the timezone formatting symbol Z is applicable only to the output dateTime format. • White space is skipped over.
Time Zone	Enumerated	The value that you set for this property is used if the value that you specified for the Default DateTime Format property does not include Time Zone information. The initial value is Use Broker Locale, which causes the broker to get the information from the underlying platform. You can change this property by selecting from the list of values.
Daylight Saving Time	Check box	Select this option if the area in the Time Zone property observes daylight saving time. If it does not observe daylight saving time, do not select this option. For example, if an area is selected in Time Zone and this option is not selected, the value passed represents the time zone without the daylight saving time.

Property	Type	Meaning
Use input UTC format on output	Check box	This property applies to elements and attributes of logical type xsd:dateTime or xsd:time that have a dateTime format of I, IU, T, or TU, or that include ZZZ or ZZZU. Such elements and attributes can specify Coordinated 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. If this property is selected, and the element or attribute is copied to an output message, the UTC format is preserved into the output message and overrides the format that is implied by the dateTime format property. If this property is cleared, or the element or attribute was not copied from an input message, the UTC format in the output message is controlled solely by the dateTime format property.

General settings

Property	Type	Meaning
Output policy for missing elements	Enumerated	Controls whether the default value or null value is used on output for missing elements. Valid values are UseDefaultValue or UseNullValue.
Derive default length from logical type	Check box	If this property is selected, an unset TDS Length property attempts to derive its default value from the simple type of the element (string, binary, integer, and decimal simple logical types only). For these logical types, it applies only to the Binary, Text, Integer, External Decimal, and Packed Decimal physical types.

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 following table.

Mnemonic string	Meaning	Default value	Associated property
<EDIFACT_CS>	Component separator in EDIFACT	:	Message set and complex type or group, *Delimiter*
<EDIFACT_DS>	Data element separator in EDIFACT	+	Message set and complex type or group, *Delimiter*
<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 or group, *Tag Data Separator*
<EDIFACT_DEC_NOTATION>	Decimal notation in EDIFACT	.	Message set, *Decimal Point*
<EDIFACT_ESC_CHAR>	Escape character in EDIFACT	?	Message set, *Escape Character*
<EDIFACT_GROUP_TERM>	Tag terminator in EDIFACT	'	Message set, *Group Terminator*
<X12_GROUP_TERM>	Tag terminator in X12	!	Message set level, *Group Terminator*
<X12_DS>	Data element separator for X12	*	Message set and complex type or group, *Delimiter*
<X12_CS>	Component separator for X12	:	Message set and complex type or group, *Delimiter*

Mnemonic string	Meaning	Default value	Associated property
<HL7_CS>	Component separator in HL7	^	Message set and complex type or group, *Delimiter*
<HL7_FS>	Data element separator in HL7	|	Message set and complex type or group, *Delimiter*
<HL7_RS>	Repeating element delimiter in HL7	~	Local element and element reference, *Repeating Element Delimiter*
<HL7_SCS>	Sub-component separator in HL7	&	Message set and complex type or group, *Delimiter*

Mnemonics for control characters are shown in the following table.

Mnemonic	Hex value	Unicode	Description
<ACK>	X'06'	<U+0006>	Acknowledge
<BEL>	X'07'	<U+0007>	Bell
<BS>	X'08'	<U+0008>	Backspace
<CAN>	X'18'	<U+0018>	Cancel
<CR>	X'0D'	<U+000D>	Carriage Return
<DC1>	X'11'	<U+0011>	Device Control One
<DC2>	X'12'	<U+0012>	Device Control Two
<DC3>	X'13'	<U+0013>	Device Control Three
<DC4>	X'14'	<U+0014>	Device Control Four
<DLE>	X'10'	<U+0010>	Data Link Escape
<EM>	X'19'	<U+0019>	End of Medium
<ENQ>	X'05'	<U+0005>	Inquiry
<EOT>	X'04'	<U+0004>	End of Transmission
<ESC>	X'1B'	<U+001B>	Escape
<ETB>	X'17'	<U+0017>	End of Transmission Block
<ETX>	X'03'	<U+0003>	End of Text
<FF>	X'0C'	<U+000C>	Form Feed
<FS>	X'1C'	<U+001C>	File Separator
<GS>	X'1D'	<U+001D>	Group Separator
<GT>	X'3E'	<U+003E>	Greater Than
<HT>	X'09'	<U+0009>	Horizontal Tabulation
<LF>	X'0A'	<U+000A>	Line Feed
<LT>	X'3C'	<U+003C>	Less Than
<NAK>	X'15'	<U+0015>	Negative Acknowledge
<NUL>	X'00'	<U+0000>	Null-
<RS>	X'1E'	<U+001E>	Record Separator
<SI>	X'0F'	<U+000F>	Locking Shift Zero (Shift In)
<SO>	X'0E'	<U+000E>	Locking Shift One (Shift Out)
<SOH>	X'01'	<U+0001>	Start of Heading

Mnemonic	Hex value	Unicode	Description
<SP>	X'20'	<U+0020>	Space
<STX>	X'02'	<U+0002>	Start of Text
<SUB>	X'1A'	<U+001A>	Substitute
<SYN>	X'16'	<U+0016>	Synchronous Idle
<US>	X'1F'	<U+001F>	Unit Separator
<VT>	X'0B'	<U+000B>	Vertical Tabulation

These mnemonics were created for characters that cannot be entered into the message editor.

You can enter a mnemonic in the form `<U+NNNN>`, where `NNNN` are hexadecimal digits. None of the characters in this structure are case-sensitive. Do not enclose spaces inside the angle brackets. These numbers represent a Unicode character, not a character in the code page of the input message.

You can enter a mnemonic in the form `<0xNN>`, where `NN` are hexadecimal digits. None of the characters in this structure are case-sensitive. Do not enclose spaces inside the angle brackets. These numbers represent a raw hexadecimal byte value, not a character in the code page of the input message.

If a mnemonic is of the form `<0xNN>`, it is applied directly to the input data, and no code page conversion takes place. Otherwise, a mnemonic is applied to the data after the data has been converted into Unicode from the code page of the input data.

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 170.

Default message set property values for TDS (part 1 of 3)

Property	Messaging standard = User Defined Text	Messaging Standard = SWIFT	Messaging standard = ACORD AL3
Group Indicator	Empty	<CR><LF>:	Empty
Group Terminator	Empty	<CR><LF>-	Empty
Delimiter	Empty	<CR><LF>:	Empty
Suppress Absent Element Delimiters	End of Type	End of Type	End of Type
Tag Data Separator	Empty	:	Empty
Length of Tag	Empty	Empty	Empty
Default CCSID	367	37	367
Trim on input	No Trim	Trim Both	No Trim
Truncate on output	Cleared	Cleared	Cleared
Escape Character	Chosen - empty	Chosen - empty	Chosen - empty

Property	Messaging standard = User Defined Text	Messaging Standard = SWIFT	Messaging standard = ACORD AL3
Quote character	Not chosen	Not chosen	Not chosen
Reserved Characters	Empty	Empty	Empty
Decimal Point	.	,	.
Packed decimal positive code	C	Not applicable	Not applicable
Strict Numeric Checking	Cleared	Selected	Selected
Derive sign from logical type	Selected	Not applicable	Not applicable
Default byte order	Big Endian	Not applicable	Not applicable
Default packed decimal byte order	Big Endian	Not applicable	Not applicable
Default float format	S390	Not applicable	Not applicable
Text boolean true value	1	1	Y
Text boolean false value	0	0	N
Text boolean null value	0	0	N
Binary boolean true value	00000001	Not applicable	Not applicable
Binary boolean false value	00000000	Not applicable	Not applicable
Binary boolean null value	00000000	Not applicable	Not applicable
Derive default dateTime format from logical type	Chosen	Chosen	Chosen
Use default DateTime Format[1]	Not chosen, but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen	Not chosen, but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen	Not chosen, but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen
Start of century for 2 digit years	53	80	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	Selected	Selected	Selected
Time Zone	Use Broker Locale	Use Broker Locale	Use Broker Locale
Daylight Saving Time	Cleared	Cleared	Cleared
Use input UTC format on output	Cleared	Cleared	Cleared
Output policy for missing elements	UseDefaultValue	UseDefaultValue	UseDefaultValue

Property	Messaging standard = User Defined Text	Messaging Standard = SWIFT	Messaging standard = ACORD AL3
Derive default length from logical type	Selected	Selected	Selected

Default message set property values for TDS (part 2 of 3)

Property	Messaging standard = EDIFACT	Messaging Standard = X12	Messaging standard = TLOG
Group Indicator	Empty	Empty	Empty
Group Terminator	<EDIFACT_GROUP_TERM>	<X12_GROUP_TERM>	Empty
Delimiter	<EDIFACT_CS>	<X12_CS>	:
Suppress Absent Element Delimiters	End of Type	End of Type	End of Type
Tag Data Separator	<EDIFACT_TAGDATA_SEP>	Empty	Empty
Length of Tag	Empty	Empty	Empty
Default CCSID	367	367	367
Trim on input	Trim Both	Trim Both	No Trim
Truncate on output	Cleared	Cleared	Cleared
Escape Character	Chosen - <EDIFACT_ESC_CHAR>	Chosen - empty	Chosen - empty
Quote character	Not chosen	Not chosen	Not chosen
Reserved Characters	<EDIFACT_ESC_CHAR> <EDIFACT_TAGDATA_SEP> <EDIFACT_GROUP_TERM> <EDIFACT_CS>	Empty	Empty
Decimal Point	<EDIFACT_DEC_NOTATION>	.	.
Packed decimal positive code	Not applicable	Not applicable	Not applicable
Strict Numeric Checking	Selected	Selected	Cleared
Derive sign from logical type	Not applicable	Not applicable	Not applicable
Default byte order	Not applicable	Not applicable	Not applicable
Default packed decimal byte order	Not applicable	Not applicable	Not applicable
Default float format	Not applicable	Not applicable	Not applicable
Text boolean true value	1	1	1
Text boolean false value	0	0	0
Text boolean null value	0	0	0
Binary boolean true value	Not applicable	Not applicable	Not applicable
Binary boolean false value	Not applicable	Not applicable	Not applicable

Property	Messaging standard = EDIFACT	Messaging Standard = X12	Messaging standard = TLOG
Binary boolean null value	Not applicable	Not applicable	Not applicable
Derive default dateTime format from logical type	Chosen	Chosen	Chosen
Use default DateTime Format[1]	Not chosen, but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen	Not chosen, but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen	Not chosen, but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen
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	Selected	Selected	Selected
Time Zone	Use Broker Locale	Use Broker Locale	Use Broker Locale
Daylight Saving Time	Cleared	Cleared	Cleared
Use input UTC format on output	Cleared	Cleared	Cleared
Output policy for missing elements	UseDefaultValue	UseDefaultValue	UseDefaultValue
Derive default length from logical type	Selected	Selected	Selected

Default message set property values for TDS (part 3 of 3)

Property	Messaging standard = HL7	Messaging Standard = CSV	Messaging standard = User Defined Mixed
Group Indicator	Empty	Empty	Empty
Group Terminator	<CR>	Empty	Empty
Delimiter	<HL7_FS>	,	Empty
Suppress Absent Element Delimiters	End of Type	Never	End of Type
Tag Data Separator	<HL7_FS>	Empty	Empty
Length of Tag	Empty	Empty	Empty
Default CCSID	367	367	850
Trim on input	No Trim	No Trim	Trim Padding Chars
Truncate on output	Cleared	Cleared	Cleared
Escape Character	Chosen - empty	Not chosen	Chosen - empty
Quote character	Not chosen	Chosen - "	Not chosen
Reserved Characters	Empty	, <CR> <LF> "	Empty

Property	Messaging standard = HL7	Messaging Standard = CSV	Messaging standard = User Defined Mixed
Decimal Point	.	.	.
Packed decimal positive code	Not applicable	C	C
Strict Numeric Checking	Cleared	Cleared	Cleared
Derive sign from logical type	Not applicable	Selected	Selected
Default byte order	Not applicable	Big Endian	Big Endian
Default packed decimal byte order	Not applicable	Big Endian	Big Endian
Default float format	Not applicable	S390	S390
Text boolean true value	1	1	1
Text boolean false value	0	0	0
Text boolean null value	0	0	0
Binary boolean true value	Not applicable	00000001	00000001
Binary boolean false value	Not applicable	00000000	00000000
Binary boolean null value	Not applicable	00000000	00000000
Derive default dateTime format from logical type	Not chosen	Chosen	Chosen
Use default DateTime Format[1]	Chosen - yyyy-MM-dd'T'HH:mm:ssZZZ	Not chosen - but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen	Not chosen - but yyyy-MM-dd'T'HH:mm:ssZZZ if chosen
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	Selected	Selected	Selected
Time Zone	Use Broker Locale	Use Broker Locale	Use Broker Locale
Daylight Saving Time	Cleared	Cleared	Cleared
Use input UTC format on output	Cleared	Cleared	Cleared
Output policy for missing elements	UseDefaultValue	UseDefaultValue	UseDefaultValue
Derive default length from logical type	Selected	Selected	Selected

Default complex type/group property values for TDS (part 1 of 3)

Property	Messaging standard = User Defined Text	Messaging standard = SWIFT	Messaging standard = ACORD AL3
Data Element Separation	Fixed Length	Tagged Delimited	Fixed Length AL3
Group Indicator	Empty	<CR><LF>:	Empty
Group Terminator	Empty	<CR><LF>-	Empty
Delimiter	Empty	<CR><LF>:	not applicable
Suppress Absent Element Delimiters	End of Type	End of Type	End of Type
Observe Element Length	Selected	Cleared	Selected
Tag Data Separator	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

Default complex type/group property values for TDS (part 2 of 3)

Property	Messaging standard = EDIFACT	Messaging standard = X12	Messaging standard = TLOG
Data Element Separation	All Elements Delimited	All Elements Delimited	Fixed length
Group Indicator	Empty	Empty	Empty
Group Terminator	<EDIFACT_GROUP_TERM>	<X12_GROUP_TERM>	Empty
Delimiter	<EDIFACT_CS>	<X12_CS>	:
Suppress Absent Element Delimiters	End of Type	End of Type	End of Type
Observe Element Length	Cleared	Cleared	Cleared
Tag Data Separator	<EDIFACT_TAGDATA_SEP>	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

Default complex type/group property values for TDS (part 3 of 3)

Property	Messaging standard = HL7	Messaging standard = CSV	Messaging standard = User Defined Mixed
Data Element Separation	All Elements Delimited	All Elements Delimited	Fixed Length
Group Indicator	Empty	Empty	Empty
Group Terminator	<CR>	Empty	Empty

Property	Messaging standard = HL7	Messaging standard = CSV	Messaging standard = User Defined Mixed
Delimiter	<HL7_FS>	,	Empty
Suppress Absent Element Delimiters	End of Type	Never	End of Type
Observe Element Length	Cleared	Cleared	Selected
Tag Data Separator	<HL7_FS>	not applicable	Empty
Length of Tag	Empty	not applicable	Empty
Length of Encoded Length	not applicable	not applicable	not applicable
Extra Chars in Encoded Length	not applicable	not applicable	not applicable

XML Wire Format message set properties

The following tables define the properties for the XML Wire Format for the message set.

Namespace settings

Property	Type	Meaning
Namespace URI	String	Enter the name of the namespace that you are using for the associated prefix.
Prefix	String	Enter the prefix to associate the element and attribute names that you use it with to the namespace name.

Namespace schema locations

Property	Type	Meaning
Namespace URI	String	Enter the namespace name that identifies which namespace you are using.
Schema location	String	Enter the location of the schema for the associated namespace name that is used to validate objects within the namespace.

XML declaration

Property	Type	Meaning
Suppress XML Declaration	Check box	Select the check box to suppress the XML declaration. If selected, the declaration (for example, `<?xml version='1.0'>`) is suppressed. By default, the check box is cleared.
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.

Property	Type	Meaning
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 is generated that is consistent with the text in the XML document. The default is Null. If the Suppress XML Declaration check box is selected, this property is ignored.
Standalone Document	Enumerated type	Select `Yes`, `No`, or `Null` from the list of values. If you select `Null`, 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. The default value is `Null`. The setting of this property does not determine whether an external DTD subset is loaded; external DTD subsets are never loaded in this release. If the Suppress XML Declaration check box is selected, this property is ignored.
Output Namespace Declaration	Enumerated type	The *Output Namespace Declaration* property controls where the namespace declarations are placed in the output XML document. Select from: • `At start of document`. Declarations for all of the entries in the *Namespace schema locations* table above are output as attributes of the message in the output XML document. The disadvantage of this option is that, in some cases, unnecessary declarations might be output. • `As required`. Declarations are output only when required by an element or attribute that is in that namespace. The disadvantage of this option is that the same namespace declaration might need to be output more than once in the output XML document. The default option is `At start of document`. This property is active only if namespaces are enabled for this message set.

XML document type settings

Property	Type	Meaning
Suppress DOCTYPE	Check box	If you select the check box, the DOCTYPE (DTD) declaration is suppressed. By default, the check box is selected.
DOCTYPE System ID	String	Specify the System ID for DOCTYPE external DTD subset (if DOCTYPE is present). This is typically set to the name of the generated (or imported) DTD for a message set. If *Suppress DOCTYPE* is set, this property is ignored and cannot be changed (the field is disabled). The default value is `www.mrmnames.net/`, followed by the message set identifier.
DOCTYPE Public ID	String	Specify the Public ID for DOCTYPE external DTD subset (if DOCTYPE is present, and System ID is specified). If *Suppress DOCTYPE* is set, this property is ignored and cannot be changed (the field is disabled). The default value is the message set identifier.

Property	Type	Meaning
DOCTYPE Text	String	Use this property to add additional DTD declarations. It is not parsed by the XML parser and, therefore, it might not be valid XML. You can include ENTITY definitions or internal DTD declarations. It is a string (up to 32 KB) in which new line and tab characters are replaced by `\n` and `\t` respectively. 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. If you have set *Suppress DOCTYPE*, this property is ignored and cannot be changed (the field is disabled). For more information, see "MRM XML: In-line DTDs and the DOCTYPE text property" on page 184. The default value is empty (not set).

XML representation of Boolean values

Property	Type	Meaning
Boolean True Value	String	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. The default is `true`. `1` is also valid.
Boolean False Value	String	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. The default is `false`. `0` is also valid.

XML representation of null values

Property	Type	Meaning
Encoding Numeric Null	Enumerated type	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 list shown: • `NULLEmpty`. If the element value is the empty string, the element is null. This is the default value. • `NULLValue`. If the element value matches that provided by associated property *Encoding Numeric Null Value*, the element is null. • `NULLXMLSchema`. If the element contains an *xsi:nil* attribute that evaluates to true, the element is null. • `NULLValueAttribute`. This option is valid only for elements that have XML Wire Format property *Render* set to either `XMLElementAttrVal` or `XMLElementAttrIDVal`. See "XML Null handling options" on page 750 for details. • `NULLAttribute` (deprecated). If the element contains an attribute with a name that matches that provided by associated property *Encoding Numeric Null Value*, and the attribute evaluates to true, the element is null. • `NULLElement` (deprecated). If the element contains a child element with a name that matches that provided by associated property *Encoding Numeric Null Value*, the element is null. See "XML Null handling options" on page 750 for full details.

Property	Type	Meaning
Encoding Numeric Null Value	String	Specify the value to qualify the *Encoding Numeric Null* property, if you have set that to `NULLValue`, `NULLAttribute`, or `NULLElement`. Refer to "XML Null handling options" on page 750 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 *Encoding Numeric Null*.
Encoding Non-Numeric Null Value	String	Specify the value to qualify the *Encoding Non-Numeric Null* property. Refer to "XML Null handling options" on page 750 for further information.

DateTime settings

Property	Type	Meaning
Derive default dateTime format from logical type	Button	Select this option if you want the default dateTime format to be determined by the logical type of the element or attribute. You can override this property for an element or attribute within a complex type.
Use default dateTime format	Button and String	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. You can override this property for an element or attribute within a complex type. For more information, see "DateTime formats" on page 782.
Start of century for 2-digit years	Integer	This property determines how 2-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 2-digit dates are interpreted as being in the range 1989 - 2088.
Days in First Week of Year	Enumerated	Specify the number of days of the new year that must fall within the first week. The start of a year typically 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; therefore, week 1 starts some days into the new year. Otherwise, it is considered to be the first week of the new year; in this case, week 1 starts some days before the new year. Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a number from the list that is displayed.
First Day Of Week	Enumerated	Specify the day on which each new week starts. Select Use Broker Locale, which causes the broker to get the information from the underlying platform, or select a value from the list that is displayed.

Property	Type	Meaning
Strict DateTime Checking	Check box	Select this option if you want to restrict dateTimes to a valid dateTime format. If Strict DateTime Checking is selected, receiving an incorrect dateTime causes an error. **Strict dateTime checking** Examples of strict dateTime checking are: • DateTimes are restricted to valid dateTimes only. When you use this option, a date such as the 35th March is not processed as 4th April, and 10:79 is not processed as 11:19. Receiving an out-of-band dateTime, such as these examples, causes an error to occur. • 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 that you require. The maximum number of digits that is permitted becomes the upper bound for a particular symbol. For example, day in month has an upper bound of 31; therefore, a format string of 'd' allows the values 2 and 21 to be parsed, but does not allow the values 32 and 210. On output, numbers are padded with zeros to the specified length. A year is a special case; see the message set property 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. • White space is not skipped over. The white space in the input string must correspond with the same number and position of white space in the formatting string. • If data remains that is not parsed in the input string after all the symbols in the formatting string have been matched, an error occurs. **Lenient dateTime checking** Examples of lenient dateTime checking are: • The parser converts out-of-band dateTime values to the appropriate in-band value. For example, a date of 2005-05-32 is converted to 2005-06-01. • Output of dateTimes always adheres 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. Therefore, dates that are not valid - for example, 2005-1-123 and 2005-011-12 - can be entered. The first value of 2005-1-123 is output as the date 2005-05-03, and the second value of 2005-011-12 is output as the date 2005-11-12. • The number of the timezone formatting symbol Z is applicable only to the output dateTime format. • White space is skipped over.
Time Zone	Enumerated	The value that you set for this property is used if the value that you specified for the Default DateTime Format property does not include Time Zone information. The initial value is Use Broker Locale, which causes the broker to get the information from the underlying platform. You can change this property by selecting from the list of values.
Daylight Saving Time	Check box	Select this option if the area in the Time Zone property observes daylight saving time. If it does not observe daylight saving time, do not select this option. For example, if an area is selected in Time Zone and this option is not selected, the value passed represents the time zone without the daylight saving time.

Property	Type	Meaning
Use input UTC format on output	Check box	This property applies to elements and attributes of logical type xsd:dateTime or xsd:time that have a dateTime format of I, IU, T, or TU, or that include ZZZ or ZZZU. Such elements and attributes can specify Coordinated 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. If this property is selected, and the element or attribute is copied to an output message, the UTC format is preserved into the output message and overrides the format that is implied by the dateTime format property. If this property is cleared, or the element or attribute was not copied from an input message, the UTC format in the output message is controlled solely by the dateTime format property.

xsi:type settings

Property	Type	Meaning
Output policy for xsi:type attributes	Enumerated type	When writing XML documents, use this property to specify the circumstances under which the xsi:type attribute of elements is produced as output. **Never** Do not produce xsi:type attributes for elements, even if xsi:type attributes appear in the message tree. **When present** Produce xsi:type attributes for elements only when xsi:type attributes appear in the message tree. This value is the default value. **Always (Simple elements only)** Ensure that all simple elements are produced with an xsi:type attribute. If a simple element already has an xsi:type attribute in the message tree, it is used; otherwise, an xsi:type attribute is generated by using the rules in the following table. **Always (All elements)** Ensure that all elements are produced with an xsi:type attribute if possible to do so. If an element already has an xsi:type attribute in the message tree, it is used; otherwise, an xsi:type attribute is generated by using the rules in the following table. **Follow SOAP Encoding rules** Follow the same behavior as for Always (Simple elements only). Additionally, produce a SOAP encoding-style attribute in the root tag of all messages.

If an xsi:type attribute needs to be produced as output, but does not appear in the message tree, the value is generated as described in the following table.

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.

Element type	Value generated when element is defined in model	Value generated when element is self-defining
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 produced.	No xsi:type attribute is produced.

Deprecated

Note: The following properties are used to control behavior of the MRM parser; they should not be changed from their default settings. These properties will be withdrawn in a future release.

Property	Type	Meaning
Root Tag Name	String	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. The default value is blank.
Suppress Timestamp Comment	Check box	If selected, the timestamp comment string in the XML output is suppressed. If not selected, the comment is *not* suppressed, and a comment of the form `<!--MRM Generated XML Output on: Tue Apr 23 09:34:42 2002-->` is included in the output message. The default is for the check box to be selected.
Enable Versioning Support	Check box	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. 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 should ensure that this check box is not selected.

MRM XML: 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*. The parser takes certain actions when constructing an output message.

1. If the output message has to be regenerated, for example if you configure a Compute node to create an 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 the output message does not have 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.

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 message 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.

Message definition file properties

The properties of a message definition file.

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 and Enumerated type	The default value for the block attribute for all complex types and elements within this message definition file.
Default final	String and Enumerated type	The default value for the final attribute for all complex types and elements within this message definition file.

Property	Type	Meaning
Use xml.xsd schema	Check box	Select this check box if you need to use the xml.xsd schema. When you select this check box, the http://www.w3.org/2001/xml.xsd schema is imported and you can use any of the constructs in that schema.

Note: The full text that describes this check box is *Use http://www.w3.org/2001/xml.xsd schema*.

Message definition file includes properties

The location of each message definition file that has been included in this message definition file is displayed.

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

The file imports properties of a message definition.

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.

Redefines are not supported, and result in a validation error. If you right-click the error message and select **Quick Fix**, you can convert the redefines construct into an include construct, which also removes the error message.

Documentation properties for all message set objects

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Message category properties

A message category provides a way of grouping your messages.

The following table describes the properties that are associated with a message category:

Property	Type	Meaning
Category Kind	Enumerated type	This property describes the purpose of the message category. Choose from the following values: • wsdl. This value is the default. Choose this value if the message category is to participate in the generation of WSDL documents. When the WSDL document is generated, the name of the message category provides the name for the <wsdl:operation> element that is generated for eligible messages in the message category. **Note:** Message categories are no longer necessary for the generation of WSDL documents; they were necessary in Version 6.0. • other. This value ndicates that the category represents a generic grouping of messages as an aid to organizing them in your workspace.
Category Usage	Enumerated type	Use this property to describe the operation type for a WSDL operation. Choose from the following values: • wsdl:request-response. This is the default if Category Kind is wsdl. • wsdl:solicit-response. • wsdl:one-way. • wsdl:notification. • empty string. This is the default if Category Kind is other.
Documentation	String	Use this property to add information to enhance the understanding of an object's function. This property is a string field; any standard alphanumeric characters can be used. If Category Kind is wsdl, the value of the field is included in any generated WSDL as the wsdl:documentation child of the operation element in the WSDL portType. If Category Kind is other, the value of the field merely documents the Message Category within your workspace.

Message category member properties

This describes the properties that are associated with a message category member.

Property	Type	Meaning
Role Name	String	If Category Kind is wsdl, the value of the property becomes the WSDL message part name and must be unique within the category. It always defaults to the message name. If Category Kind is other, the value of the property has no particular significance.
Role Type	Enumerated type	This property determines the role that the message plays in the message category. Select from: • `wsdl:input` • `wsdl:output` • `wsdl:return` • `wsdl:fault` • `empty string` If Category Kind is wsdl, the default value is wsdl:input. This property 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 that is used as the return value and in this instance can be omitted from the parameterOrder attribute. No more than one message can have Role Type of wsdl:return. If Category Kind is other, the value defaults to an empty string and this property has no role in the message category.
Role Usage	Enumerated type	This property determines the role that the message plays in the SOAP binding. Select from: • `soap:body` • `soap:header` • `soap:fault` • `soap:headerfault` • `empty string` If Category Kind is wsdl, this property defaults to soap:body and dictates the SOAP-binding child of the WSDL input, output, or fault element. If Category Kind is other, this property is deactivated.
Documentation	String	This is a string property; any standard alphanumeric characters can be used. If Category Kind is wsdl, the value of the property is included in any generated WSDL as the wsdl:documentation child of the WSDL input, output, or fault element under the WSDL portType. If Category Kind is other, the value merely documents the Message Category within your workspace.

Message model object properties

Access property information by property kind, or by object.

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 218
- "Documentation properties for all message set objects" on page 187

Alternatively, you can access the property information by object, starting from:

- "Message model object properties by object" on page 252

Deprecated objects are dealt with separately. For further information, see "Deprecated message model object properties" on page 615

Logical properties for message model objects

Logical property information is available for certain objects.

- "Attribute group reference logical properties"
- "Attribute reference logical properties"
- "Complex type logical properties" on page 190
- "Element reference logical properties" on page 195
- "Global attribute logical properties" on page 196
- "Global attribute group logical properties" on page 199
- "Global element logical properties" on page 200
- "Global group logical properties" on page 203
- "Group reference logical properties" on page 205
- "Key logical properties" on page 205
- "Keyref logical properties" on page 205
- "Local element logical properties" on page 209
- "Local group logical properties" on page 213
- "Message logical properties" on page 215
- "Simple type logical properties" on page 215
- "Unique logical properties" on page 216
- "Wildcard attribute logical properties" on page 217
- "Wildcard element logical properties" on page 217

Attribute group reference logical properties

The logical properties of an attribute group reference.

Property	Type	Meaning
Reference Name	Enumerated type	The Reference Name is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Attribute reference logical properties

The logical properties of an attribute reference.

Property	Type	Meaning
Reference Name	Enumerated type	The Reference Name is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Property	Type	Meaning
Usage	Enumerated type	Use this property with the Value property found in an attribute object. The default value for the Usage property is optional. Select from the following options: • optional. – If the Value property is set to default, and no data has been entered in the Value property, the attribute can appear once and can have any value. – If the Value property is set to default, the attribute can appear once. If it does not appear, its value is the data that has been entered in the Value property. If it does appear, it is the value given. – Where the Value 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 Value property. If it does not appear, its value is the data that has been entered in the Value property. • prohibited. The attribute must not appear. • required. – If the Value property is set to default, and no data has been entered in the Value property, the attribute must appear once and can have any value. – If the Value property is set to fixed, the attribute must appear once, and it must match the data that has been entered in the Value property.

Complex type logical properties

The logical properties of a complex type include properties that describe content and substitution settings.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.
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.

Property	Type	Meaning
Derived By	Enumerated type	If this property is active, select one of the following options: • restriction. If a complex type is derived by restriction, the content model of the complex type is a subset of the base type. • 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. Derivation by list or union is not supported.

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 be edited only 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 "MRM content validation" on page 194.

Property	Type	Meaning
Local Group	Button	Select this property if the content of your complex type is a local group.

Property	Type	Meaning
Composition	Enumerated type	Define the order, and the number of occurrences, of the elements and groups in your messages. Composition does not affect the attributes in a complex type. Select one of the following options: • Empty • sequence. If you select this option, you can define members that are elements or groups. These members, if present, must appear in the specified order in the message. They can repeat, and the same element or group can appear more than once. • choice. If you select this option, you can define members that are elements or groups. Exactly one of the defined members must be present in the message, and can repeat. 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. • all. If you select this option, you can define members that are elements; groups are not allowed. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can be used only at the top level of a complex type; it cannot be a member of another group within a type. • unorderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements can appear in any order in the message. • orderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements must appear in the specified order in the message. • message. This option is supported only by the MRM domain.If you select this option, you can define only messages as members. Each member can repeat, but the same message cannot appear twice in the list of members. Like choice, only one of the defined members can be present in a message. Unlike choice, when writing a message, if the complex type or group has more than one member, the bit stream is not padded to the length of the longest member. 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 26.

Property	Type	Meaning
Content Validation	Enumerated type	Content Validation is used only by the MRM domain. If validation is enabled in your message flow, Content Validation specifies the strictness of the MRM validation for members of a complex type or group. See “MRM content validation” on page 194 for further details. Select from the following options: • Closed. The complex type can only contain the child elements that you have added to it. • Open Defined. The complex type can contain any valid element defined within the message set. • Open. The complex type can contain any valid element, not just those that you have added to this complex type. See “Combinations of Composition and Content Validation” on page 300 for further details of these options.
Group Reference	Button	Select this option if the content of your complex type is a reference to a group object
Group Name	Enumerated type	The Group Name is the name of the group that this complex type refers to. The groups available to be referenced can be selected from the drop down list.
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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.
Mixed	Check box	Select this option when the complex type has mixed content, and contains character data and sub-elements.

Substitution settings

Property	Type	Meaning
Final	Multiple selection enumerated type	The final attribute on a complex type controls whether other types can be derived from it. Valid values are extension/restriction/all. You can select from one or more of the following: • Empty • `restriction`. Prohibit type substitution by elements whose types are restrictions of the head element's type. • `extension`. Prohibit type substitution by elements whose types are extensions of the head element's type. • `#all`. Prohibit substitution by any method. To select more than one, you will need to type the selection into the property field.

Property	Type	Meaning
Block	Multiple selection enumerated type	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: • Empty • `restriction`. Prohibit type substitution by elements whose types are restrictions of the head element's type. • `extension`. Prohibit type substitution by elements whose types are extensions of the head element's type. • `#all`. Prohibit substitution by any method. To select more than one, you will need to type the selection into the property field.
Abstract	Check box	If selected, no elements based on this type can appear in the message.

MRM content validation:

Content Validation is applied when the domain is MRM and validation is enabled. The Content Validation property specifies how strictly the MRM parser validates the members of a complex type or group.

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 30 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 30 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 cannot 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

The logical properties of an element reference include properties that specify the number of occurrences of the element reference.

Property	Type	Meaning
Reference Name	Enumerated type	The Reference Name is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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 element's Value properties. The following table 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 Value property. In this example, the element must contain the text Delta.
2	-1	Delta		The element must appear twice or more, and it must match the data that has been entered in the Value property. In this example, at least two elements must contain the text Delta.
0	1			The element is optional, can appear once, and can have any value.

Min Occurs	Max Occurs	Fixed	Default	Notes
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 Value property. If it does not appear, its value is the data that has been entered in the Value property.
0	1		Delta	The element is optional, and can appear once. If it does not appear, its value is the data that has been entered in the Value 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 Value property, else it is the value given in the element.
0	0			The element is prohibited, and must not appear.

Global attribute logical properties

The logical properties of a global attribute.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.

Property	Type	Meaning
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • int • string • Boolean • hexBinary • dateTime • date • time • decimal • float • (More...) • (New Simple Type) • (New Complex Type) If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.
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 are displayed in the drop-down list.

Value

The *Value* properties are used with the *Usage* property in an Attribute Reference or a Local Attribute.

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.

Property	Type	Meaning
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. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds 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 remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Global attribute group logical properties

The logical properties of an attribute group.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.

Global element logical properties

The logical properties of a global element.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • `int` • `string` • `Boolean` • `hexBinary` • `dateTime` • `date` • `time` • `decimal` • `float` • `(More...)` • `(New Simple Type)` • `(New Complex Type)` If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.

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 are displayed in the drop-down list.

Value

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.

Property	Type	Meaning
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.
Nillable	Check box	Select this option if you want the element to be able to be defined as null. A null value is distinct from being empty, when the element contains no data.
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. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds 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 remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

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	Limit the set of elements that can belong to its substitution group. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element. • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element. • #all. Prohibit substitution by all methods.
Block	Enumerated type	Limit the set of elements that can be substituted for this element in a message. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element • substitution. Prohibit element substitution by members of the element's substitution group. • #all. Prohibit substitution by all methods.
Substitution Group	Enumerated type	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 option 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

The logical properties of a global group.

Valid children in a global group that depend on both **Composition** and **Content Validation** are shown in "MRM content validation" on page 194.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.

Property	Type	Meaning
Composition	Enumerated type	Define the order, and the number of occurrences, of the elements and groups in your messages. Composition does not affect the attributes in a complex type. Select from the following options: • Empty • sequence. If you select this option, you can define members that are elements or groups. These members, if present, must appear in the specified order in the message. They can repeat, and the same element or group can appear more than once. • choice. If you select this option, you can define members that are elements or groups. Exactly one of the defined members must be present in the message, and can repeat. 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. • all. If you select this option, you can define members that are elements; groups are not allowed. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can be used only at the top level of a complex type; it cannot be a member of another group within a type. • unorderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements can appear in any order in the message. • orderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements must appear in the specified order in the message. • message. This option is supported only by the MRM domain. If you select this option, you can define only messages as members. Each member can repeat, but the same message cannot appear twice in the list of members. Like choice, only one of the defined members can be present in a message. Unlike choice, when writing a message, if the complex type or group has more than one member, the bit stream is not padded to the length of the longest member. 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 26.

Property	Type	Meaning
Content Validation	Enumerated type	Content Validation is used only by the MRM domain. If validation is enabled in your message flow, Content Validation specifies the strictness of the MRM validation for members of a complex type or group. See "MRM content validation" on page 194 for further details. Select from the following options: • Closed. The complex type can only contain the child elements that you have added to it. • Open Defined. The complex type can contain any valid element defined within the message set. • Open. The complex type can contain any valid element, not just those that you have added to this complex type. See "Combinations of Composition and Content Validation" on page 300 for further details of these options.

Group reference logical properties

The logical properties of a group reference include properties that specify the number of occurrences of the group reference.

Property	Type	Meaning
Reference Name	Enumerated type	The Reference Name is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Occurrence properties

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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.

Key logical properties

There are no properties to show.

Keyref logical properties

This describes the logical properties of a keyref.

There are no properties to show.

Local attribute logical properties

The logical properties of a local attribute.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • `-` the hyphen • `_` the underscore • `.` the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • `int` • `string` • `Boolean` • `hexBinary` • `dateTime` • `date` • `time` • `decimal` • `float` • `(More...)` • `(New Simple Type)` • `(New Complex Type)` If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.

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 are displayed in the drop-down list.

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	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.

Property	Type	Meaning
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.
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. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds 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 remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Usage properties

Property	Type	Meaning
Usage	Enumerated type	Use this property with the Value property found in an attribute object. The default value for the Usage property is optional. Select from the following options: • optional. – If the Value property is set to default, and no data has been entered in the Value property, the attribute can appear once and can have any value. – If the Value property is set to default, the attribute can appear once. If it does not appear, its value is the data that has been entered in the Value property. If it does appear, it is the value given. – Where the Value 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 Value property. If it does not appear, its value is the data that has been entered in the Value property. • prohibited. The attribute must not appear. • required. – If the Value property is set to default, and no data has been entered in the Value property, the attribute must appear once and can have any value. – If the Value property is set to fixed, the attribute must appear once, and it must match the data that has been entered in the Value property.

Local element logical properties

The logical properties of a local element include properties that specify the number of occurrences and value of the local element.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.

Property	Type	Meaning
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • int • string • Boolean • hexBinary • dateTime • date • time • decimal • float • (More...) • (New Simple Type) • (New Complex Type) If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.
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 are displayed in the drop-down list.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs.

Property	Type	Meaning
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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.

Value

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.

Property	Type	Meaning
Nillable	Check box	Select this option if you want the element to be able to be defined as null. A null value is distinct from being empty, when the element contains no data.
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. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds 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 remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

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	Limit the set of elements that can belong to its substitution group. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element. • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element. • #all. Prohibit substitution by all methods.
Block	Enumerated type	Limit the set of elements that can be substituted for this element in a message. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element • substitution. Prohibit element substitution by members of the element's substitution group. • #all. Prohibit substitution by all methods.
Substitution Group	Enumerated type	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.

Property	Type	Meaning
Abstract	Check box	Select this option 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

The logical properties of a local group include properties that specify the number of occurrences of the local group.

Valid children in a local group that depend on both **Composition** and **Content Validation** are shown in "MRM content validation" on page 194.

Property	Type	Meaning
Composition	Enumerated type	Define the order, and the number of occurrences, of the elements and groups in your messages. Composition does not affect the attributes in a complex type. Select from the following options: • Empty • sequence. If you select this option, you can define members that are elements or groups. These members, if present, must appear in the specified order in the message. They can repeat, and the same element or group can appear more than once. • choice. If you select this option, you can define members that are elements or groups. Exactly one of the defined members must be present in the message, and can repeat. 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. • all. If you select this option, you can define members that are elements; groups are not allowed. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can be used only at the top level of a complex type; it cannot be a member of another group within a type. • unorderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements can appear in any order in the message. • orderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements must appear in the specified order in the message. • message. This option is supported only by the MRM domain. If you select this option, you can define only messages as members. Each member can repeat, but the same message cannot appear twice in the list of members. Like choice, only one of the defined members can be present in a message. Unlike choice, when writing a message, if the complex type or group has more than one member, the bit stream is not padded to the length of the longest member. 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 26.

Property	Type	Meaning
Content Validation	Enumerated type	Content Validation is used only by the MRM domain. If validation is enabled in your message flow, Content Validation specifies the strictness of the MRM validation for members of a complex type or group. See "MRM content validation" on page 194 for further details. Select from the following options: • Closed. The complex type can only contain the child elements that you have added to it. • Open Defined. The complex type can contain any valid element defined within the message set. • Open. The complex type can contain any valid element, not just those that you have added to this complex type. See "Combinations of Composition and Content Validation" on page 300 for further details of these options.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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.

Message logical properties

This section describes the logical properties of a message.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.
Message Alias	String	Specify an alternative unique value that identifies the message. This property is only required if you are using the MRM domain and the Message Identity technique to identify embedded messages, and the bit stream does not contain the actual message name.

Simple type logical properties

The logical properties of a simple type.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.
Base Type	Enumerated type	This property only applies to a simple type restriction. 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.

Property	Type	Meaning
Item Type	Enumerated type	This property only applies to a simple type list. You can use this property to select the type that is used as the item type of the list.
Variety	Enumerated type	This property displays the variety of the simple type you have selected, either atomic, list, or union.

A simple type can also have "Simple type logical value constraints."

Simple type logical value constraints:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong
Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Unique logical properties

There are no properties to show.

Wildcard attribute logical properties

The logical properties of a wildcard attribute.

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, Process Content defines how the attribute is validated. Select one of the following options: • strict. The parser can match only against attributes declared in the specified namespace. • lax. The parser attempts to match against attributes declared in all accessible namespaces. If the specified namespace cannot be found, an error is not generated. • skip. If you select skip, the parser does not perform validation on the attribute.

Wildcard element logical properties

The logical properties of a wildcard element include properties that specify the number of occurrences of the wildcard element.

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 element that corresponds to a wildcard in the message model, Process Content defines how the element is validated. Select one of the following options: • strict. The parser can match only against elements declared in the specified namespace. • lax. The parser attempts to match against elements declared in any accessible namespace. If the specified namespace cannot be found, an error is not generated. • skip. If you select skip the parser does not perform validation on the element.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs. This property is ignored by brokers that are earlier than WebSphere Message Broker Version 6.0.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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 earlier than WebSphere Message Broker Version 6.0.

Physical properties for message model objects

CWF, XML, and TDS format 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 224
- "TDS format physical properties for message model objects" on page 233

Custom Wire Format physical properties for message model objects

Custom wire format physical property information is available for some objects.

- "Attribute group reference CWF properties" on page 219
- "Attribute reference CWF properties" on page 219
- "Complex type CWF properties" on page 219
- "Element reference CWF properties" on page 219
- "Global attribute CWF properties" on page 220
- "Global attribute group CWF properties" on page 220
- "Global element CWF properties" on page 220
- "Global group CWF properties" on page 220
- "Group reference CWF properties" on page 220
- "Key CWF properties" on page 221
- "Keyref CWF properties" on page 221
- "Local element CWF properties" on page 222
- "Local group CWF properties" on page 223
- "Message CWF properties" on page 224

- "Simple type CWF properties" on page 224
- "Unique CWF properties" on page 224
- "Wildcard attribute CWF properties" on page 224
- "Wildcard element CWF properties" on page 224

Attribute group reference CWF properties:

There are no properties to show.

Attribute reference CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong
Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Complex type CWF properties:

There are no properties to show.

Element reference CWF properties:

The properties, and permissible values, vary according to the type of object.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong
Float types	**Integer types**	**Interval types**	**String types**
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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 CWF properties to show for a global element.

Global group CWF properties:

There are no properties to show.

Group reference CWF properties:

The CWF properties of a group reference.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* 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:

This describes the CWF properties of a keyref.

There are no properties to show.

Local attribute CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong
Float types	**Integer types**	**Interval types**	**String types**
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong

Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Local group CWF properties:

The CWF properties of a local group are described in the following tables.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* 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 some objects.

- "Unique XML properties" on page 233
- "Wildcard attribute XML properties" on page 233
- "Wildcard element XML properties" on page 233

Attribute group reference XML properties:

The XML properties of an attribute group reference.

There are no properties to show.

Attribute reference XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong
Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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, and their permissible values, vary according to the type of object.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong
Float types	**Integer types**	**Interval types**	**String types**
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong

Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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, and their permissible values, vary according to the type of object.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong

Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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.

There are no properties to show.

Group reference XML properties:

The XML properties of a group reference.

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, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong
Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong

Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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:

The following tables describe the XML properties of a message.

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. If namespaces have not been enabled, this property displays <no target namespace>. This property overrides the same property at the message set level.
Schema location	String	Enter the location of the schema for the associated namespace name to be used to validate objects in the namespace.

XML declarations

Property	Type	Meaning
Output Namespace Declaration	Enumerated type	The Output Namespace Declaration property controls where the namespace declarations are placed in the output XML document. Select from: • At start of document. Declarations for all of the entries in the Namespace schema locations table above are produced as attributes of the message in the output XML document. The disadvantage of this option is that in some cases unnecessary declarations might be produced. • As required. Declarations are produced only when required by an element or attribute that is in that namespace. The disadvantage of this option is that the same namespace declaration might need to be produced more than once in the output XML document. The default option is At start of document. This property is active only if namespaces are enabled for this message set.

XML document type settings

Property	Type	Meaning
DOCTYPE System ID	String	Specify the System ID for DOCTYPE external DTD subset. It overrides the equivalent message set property setting for this particular message. If the message set property Suppress DOCTYPE is set to Yes, this parameter is ignored and cannot be changed (the field is disabled). The default value is the value that you specified for the DOCTYPE System ID property for the message set.
DOCTYPE Public ID	String	Specify the Public ID for DOCTYPE external DTD subset. It overrides the equivalent message set property setting for this particular message. If the message set property Suppress DOCTYPE is set to Yes, this parameter is ignored and cannot be changed (the field is disabled) . The default value is the value that you specified for the DOCTYPE Public ID property for the message set.
DOCTYPE Text	String	Enter optional additional text to include within the DOCTYPE. It overrides the message set property for this particular message. If the message set property Suppress DOCTYPE is set to Yes, this parameter is ignored and cannot be changed (the field is disabled). For more information, see “MRM XML: In-line DTDs and the DOCTYPE text property” on page 184. The default value is the value that you specified for the DOCTYPE Text property for the message set.

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 Root Tag Name property for the message set. **Note:** This property is deprecated. Do not change its value from its default setting.

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 (for output) in the resulting XML document. Select one of the following values from the drop-down list: • XMLElement. 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 XML Name property to the same value, both objects must refer to the same element. This is the default value for element objects. • XMLElementAttrID. 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 ID Attribute Name and a value as specified in ID Attribute Value. If you select this value for one object, and set this same value or the value XMLElementAttrIDVal for a second object, and set XML Name, ID Attribute Name, ID Attribute Value to the same values: – You must also set Value Attribute Name to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.

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.

TDS format physical properties for message model objects

Some objects have TDS wire format properties.

TDS format physical property information is available for the following objects:

- "Wildcard attribute TDS properties" on page 252
- "Wildcard element TDS properties" on page 252

Attribute group reference TDS properties:

There are no properties to show.

Attribute reference TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong
Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Complex type TDS properties:

The TDS properties of a complex type.

Field Identification

If the complex type is based on a global group, the TDS properties listed are located in the global group. If so, any changes to these properties are applied to the global group, and affect all references to the group (including any other complex types which are based on it).

Property	Type	Meaning
Data Element Separation	Enumerated type	Select one of the following values to specify the method that is used to separate the data elements within the type. • Tagged Delimited. This value indicates that all elements within the complex type are identified by a tag, and, if a value is specified in the optional Delimiter property, are separated by that value. You must set the Tag property for all child elements of simple type, and you can set the Delimiter property to a non-empty value. See "Global element TDS properties" on page 240. You must also set the Tag Data Separator or the Length of Tag property. • 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 Tag property for each of the child elements of this complex type, and each child element must have a Length or Length Reference property assigned to it. You must also set the Tag Data Separator or the Length of Tag property. • 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 the Length of Tag property, or variable length delimited by the Tag Data Separator property. You must also set the Length Of Encoded Length property so that the parser knows the size of the length field, and set the Extra Chars in Encoded Length property. This property tells the parser what to subtract from the value in the Length Of Encoded Length property to get the actual length of the data that follows the length field. This method provides a more flexible way of handling ACORD AL3 standard messages than using the Fixed Length AL3 value, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard. • All Elements Delimited. This value indicates that all elements within the complex type are separated by a delimiter. You must set a value in the Delimiter property. • Variable Length Elements Delimited. This value indicates that some of the elements within the complex type might be of variable length. Variable length elements must be delimited by the value specified in the Delimiter property. • Use Data Pattern. This value indicates that the parser determines the elements by matching the data with the regular expression that is set in the Data Pattern property of the element or type member. See "Message definition file properties" on page 185. • 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 Length property to the offset. See "Global element TDS properties" on page 240. If you set the Data Element Separation property of a complex type to Fixed Length, you must also set the Data Element Separation property of all complex children of this type to Fixed Length. Each child element must have a Length or Length Reference property assigned to it. • 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 regarding missing optional elements, encoded lengths, and versioning, must be applied. If you set the Data Element Separation property of a complex type to Fixed Length AL3, you must also set the Data Element Separation property of all complex children of this type to Fixed Length AL3. • Undefined. This value is set automatically if you set the Type Composition property of a complex type to Message, and you cannot change it to any other value. Do not set the Type Composition property to Empty, Choice, Unordered Set, Ordered Set, Sequence, or Simple Unordered Set. If you do, you cannot check in the type.

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character, or string, that precedes the data that belongs to a group, or a complex type, within the bit stream.
Group Terminator	String	Specify the value of a special character, or string, that terminates the data that belongs 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 that is used between data elements. This property applies only to the delimited Data Element Separation methods (Tagged Delimited, All Elements Delimited, and Variable Length Elements Delimited).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select whether you want delimiters to be suppressed for elements that are missing within a message. Select from: • 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 three elements and only two are present, the last delimiter can be omitted from the message. • Never. Use this option to ensure that even if optional elements are not present, all delimiters are written out. Use this option when the same delimiter is used to delimit parent and child objects. For example, if an optional child element is missing, message processing applications cannot tell where the child elements in a message end and the next parent element starts, if the delimiters are all the same.
Observe Element Length	Check box	This property is applicable when Data Element Separation is All Elements Delimited or Tagged Delimited. Select this check box if the Length property of child simple elements is significant when parsing and writing. • During parsing, an exception is thrown if the length of the extracted data exceeds the specified length. Otherwise, the data is trimmed according to the Justification and Padding Character properties of the child element. • During writing, an exception is thrown if the data to write exceeds the specified length. Otherwise, the data is padded according to the Justification and Padding Character properties of the child element. Clear this check box to ignore the Length property when parsing and writing. The default value depends on the setting of the Messaging Standard property (at the message set level) and the Data Element Separation property. • If Data Element Separation is All Elements Delimited and the Messaging Standard is TLOG, the check box is selected. • If Data Element Separation is All Elements Delimited and the Messaging Standard is other than TLOG, the check box is cleared. • If Data Element Separation is Tagged Delimited, the check box is cleared. For all other data element separation methods, the check box is disabled and does not influence the behavior of the TDS parser.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The Tag Data Separator and Length of Tag properties are mutually exclusive. If you set the property Tag Data Separator, it overrides Length of Tag. This property applies only to the tagged Data Element Separation methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).

Property	Type	Meaning
Length of Tag	Button and Integer	Specify the length of a tag value. When the message is parsed, this property allows tags to be extracted from the bit stream if the Tag Data Separator property is not set. The Tag Data Separator and Length of Tag properties are mutually exclusive. If you set the property Tag Data Separator, it overrides this value. This property applies only to the tagged Data Element Separation methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).
Length of Encoded Length	Integer	Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647. You must set this property if you have set the Data Element Separation property to Tagged Encoded Length; it is not valid otherwise. The actual number of data characters that are parsed depends on the value of the Extra Chars in Encoded Length property.
Extra Chars in Encoded Length	Integer	(Only valid if the Data Element Separation 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.) 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. You must set this property if you have set the Data Element Separation property to Tagged Encoded Length, and the actual number of data characters is less than the value found in the length field.

Element reference TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	Complex types
- base64Binary - hexBinary	- Boolean	
DateTime types	Decimal types	Float types
- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong	- double - float

Integer types	Interval types	String types
- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong

Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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:

The TDS properties of a global attribute group.

There are no properties to show.

Global element TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	Complex types
- base64Binary - hexBinary	- Boolean	
DateTime types	**Decimal types**	**Float types**
- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong	- double - float

Integer types	Interval types	String types
- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Global group TDS properties:

The TDS properties of a global group.

Field Identification

Property	Type	Meaning
Data Element Separation	Enumerated type	Select one of the following values to specify the method that is used to separate the data elements within the type. • Tagged Delimited. This value indicates that all elements within the complex type are identified by a tag, and, if a value is specified in the optional Delimiter property, are separated by that value. You must set the Tag property for all child elements of simple type, and you can set the Delimiter property to a non-empty value. See "Global element TDS properties" on page 240. You must also set the Tag Data Separator or the Length of Tag property. • 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 Tag property for each of the child elements of this complex type, and each child element must have a Length or Length Reference property assigned to it. You must also set the Tag Data Separator or the Length of Tag property. • 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 the Length of Tag property, or variable length delimited by the Tag Data Separator property. You must also set the Length Of Encoded Length property so that the parser knows the size of the length field, and set the Extra Chars in Encoded Length property. This property tells the parser what to subtract from the value in the Length Of Encoded Length property to get the actual length of the data that follows the length field. This method provides a more flexible way of handling ACORD AL3 standard messages than using the Fixed Length AL3 value, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard. • All Elements Delimited. This value indicates that all elements within the complex type are separated by a delimiter. You must set a value in the Delimiter property. • Variable Length Elements Delimited. This value indicates that some of the elements within the complex type might be of variable length. Variable length elements must be delimited by the value specified in the Delimiter property. • Use Data Pattern. This value indicates that the parser determines the elements by matching the data with the regular expression that is set in the Data Pattern property of the element or type member. See "Message definition file properties" on page 185. • 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 Length property to the offset. See "Global element TDS properties" on page 240. If you set the Data Element Separation property of a complex type to Fixed Length, you must also set the Data Element Separation property of all complex children of this type to Fixed Length. Each child element must have a Length or Length Reference property assigned to it. • 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 regarding missing optional elements, encoded lengths, and versioning, must be applied. If you set the Data Element Separation property of a complex type to Fixed Length AL3, you must also set the Data Element Separation property of all complex children of this type to Fixed Length AL3. • Undefined. This value is set automatically if you set the Type Composition property of a complex type to Message, and you cannot change it to any other value. Do not set the Type Composition property to Empty, Choice, Unordered Set, Ordered Set, Sequence, or Simple Unordered Set. If you do, you cannot check in the type.

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character, or string, that precedes the data that belongs to a group, or a complex type, within the bit stream.
Group Terminator	String	Specify the value of a special character, or string, that terminates the data that belongs 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 that is used between data elements. This property applies only to the delimited Data Element Separation methods (Tagged Delimited, All Elements Delimited, and Variable Length Elements Delimited).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select whether you want delimiters to be suppressed for elements that are missing within a message. Select from: • 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 three elements and only two are present, the last delimiter can be omitted from the message. • Never. Use this option to ensure that even if optional elements are not present, all delimiters are written out. Use this option when the same delimiter is used to delimit parent and child objects. For example, if an optional child element is missing, message processing applications cannot tell where the child elements in a message end and the next parent element starts, if the delimiters are all the same.
Observe Element Length	Check box	This property is applicable when Data Element Separation is All Elements Delimited or Tagged Delimited. Select this check box if the Length property of child simple elements is significant when parsing and writing. • During parsing, an exception is thrown if the length of the extracted data exceeds the specified length. Otherwise, the data is trimmed according to the Justification and Padding Character properties of the child element. • During writing, an exception is thrown if the data to write exceeds the specified length. Otherwise, the data is padded according to the Justification and Padding Character properties of the child element. Clear this check box to ignore the Length property when parsing and writing. The default value depends on the setting of the Messaging Standard property (at the message set level) and the Data Element Separation property. • If Data Element Separation is All Elements Delimited and the Messaging Standard is TLOG, the check box is selected. • If Data Element Separation is All Elements Delimited and the Messaging Standard is other than TLOG, the check box is cleared. • If Data Element Separation is Tagged Delimited, the check box is cleared. For all other data element separation methods, the check box is disabled and does not influence the behavior of the TDS parser.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The Tag Data Separator and Length of Tag properties are mutually exclusive. If you set the property Tag Data Separator, it overrides Length of Tag. This property applies only to the tagged Data Element Separation methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).

Property	Type	Meaning
Length of Tag	Button and Integer	Specify the length of a tag value. When the message is parsed, this property allows tags to be extracted from the bit stream if the Tag Data Separator property is not set. The Tag Data Separator and Length of Tag properties are mutually exclusive. If you set the property Tag Data Separator, it overrides this value. This property applies only to the tagged Data Element Separation methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).
Length of Encoded Length	Integer	Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647. You must set this property if you have set the Data Element Separation property to Tagged Encoded Length; it is not valid otherwise. The actual number of data characters that are parsed depends on the value of the Extra Chars in Encoded Length property.
Extra Chars in Encoded Length	Integer	(Only valid if the Data Element Separation 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.) 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. You must set this property if you have set the Data Element Separation property to Tagged Encoded Length, and the actual number of data characters is less than the value found in the length field.

Group reference TDS properties:

The following tables describe the TDS properties of a group reference.

Field identification

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Occurrences

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.
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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.

Key TDS properties:

There are no properties to show.

Keyref TDS properties:

There are no properties to show.

Local attribute TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong

Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	Complex types
- base64Binary - hexBinary	- Boolean	
DateTime types	**Decimal types**	**Float types**
- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong	- double - float

Integer types	Interval types	String types
- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

Local group TDS properties:

TDS properties of a local group.

Field Identification

Property	Type	Meaning
Data Element Separation	Enumerated type	Select one of the following values to specify the method that is used to separate the data elements within the type. • Tagged Delimited. This value indicates that all elements within the complex type are identified by a tag, and, if a value is specified in the optional Delimiter property, are separated by that value. You must set the Tag property for all child elements of simple type, and you can set the Delimiter property to a non-empty value. See "Global element TDS properties" on page 240. You must also set the Tag Data Separator or the Length of Tag property. • 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 Tag property for each of the child elements of this complex type, and each child element must have a Length or Length Reference property assigned to it. You must also set the Tag Data Separator or the Length of Tag property. • 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 the Length of Tag property, or variable length delimited by the Tag Data Separator property. You must also set the Length Of Encoded Length property so that the parser knows the size of the length field, and set the Extra Chars in Encoded Length property. This property tells the parser what to subtract from the value in the Length Of Encoded Length property to get the actual length of the data that follows the length field. This method provides a more flexible way of handling ACORD AL3 standard messages than using the Fixed Length AL3 value, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard. • All Elements Delimited. This value indicates that all elements within the complex type are separated by a delimiter. You must set a value in the Delimiter property. • Variable Length Elements Delimited. This value indicates that some of the elements within the complex type might be of variable length. Variable length elements must be delimited by the value specified in the Delimiter property. • Use Data Pattern. This value indicates that the parser determines the elements by matching the data with the regular expression that is set in the Data Pattern property of the element or type member. See "Message definition file properties" on page 185. • 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 Length property to the offset. See "Global element TDS properties" on page 240. If you set the Data Element Separation property of a complex type to Fixed Length, you must also set the Data Element Separation property of all complex children of this type to Fixed Length. Each child element must have a Length or Length Reference property assigned to it. • 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 regarding missing optional elements, encoded lengths, and versioning, must be applied. If you set the Data Element Separation property of a complex type to Fixed Length AL3, you must also set the Data Element Separation property of all complex children of this type to Fixed Length AL3. • Undefined. This value is set automatically if you set the Type Composition property of a complex type to Message, and you cannot change it to any other value. Do not set the Type Composition property to Empty, Choice, Unordered Set, Ordered Set, Sequence, or Simple Unordered Set. If you do, you cannot check in the type.

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character, or string, that precedes the data that belongs to a group, or a complex type, within the bit stream.
Group Terminator	String	Specify the value of a special character, or string, that terminates the data that belongs 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 that is used between data elements. This property applies only to the delimited Data Element Separation methods (Tagged Delimited, All Elements Delimited, and Variable Length Elements Delimited).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select whether you want delimiters to be suppressed for elements that are missing within a message. Select from: • 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 three elements and only two are present, the last delimiter can be omitted from the message. • Never. Use this option to ensure that even if optional elements are not present, all delimiters are written out. Use this option when the same delimiter is used to delimit parent and child objects. For example, if an optional child element is missing, message processing applications cannot tell where the child elements in a message end and the next parent element starts, if the delimiters are all the same.
Observe Element Length	Check box	This property is applicable when Data Element Separation is All Elements Delimited or Tagged Delimited. Select this check box if the Length property of child simple elements is significant when parsing and writing. • During parsing, an exception is thrown if the length of the extracted data exceeds the specified length. Otherwise, the data is trimmed according to the Justification and Padding Character properties of the child element. • During writing, an exception is thrown if the data to write exceeds the specified length. Otherwise, the data is padded according to the Justification and Padding Character properties of the child element. Clear this check box to ignore the Length property when parsing and writing. The default value depends on the setting of the Messaging Standard property (at the message set level) and the Data Element Separation property. • If Data Element Separation is All Elements Delimited and the Messaging Standard is TLOG, the check box is selected. • If Data Element Separation is All Elements Delimited and the Messaging Standard is other than TLOG, the check box is cleared. • If Data Element Separation is Tagged Delimited, the check box is cleared. For all other data element separation methods, the check box is disabled and does not influence the behavior of the TDS parser.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The Tag Data Separator and Length of Tag properties are mutually exclusive. If you set the property Tag Data Separator, it overrides Length of Tag. This property applies only to the tagged Data Element Separation methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).

Property	Type	Meaning
Length of Tag	Button and Integer	Specify the length of a tag value. When the message is parsed, this property allows tags to be extracted from the bit stream if the Tag Data Separator property is not set. The Tag Data Separator and Length of Tag properties are mutually exclusive. If you set the property Tag Data Separator, it overrides this value. This property applies only to the tagged Data Element Separation methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).
Length of Encoded Length	Integer	Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647. You must set this property if you have set the Data Element Separation property to Tagged Encoded Length; it is not valid otherwise. The actual number of data characters that are parsed depends on the value of the Extra Chars in Encoded Length property.
Extra Chars in Encoded Length	Integer	(Only valid if the Data Element Separation 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.) 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. You must set this property if you have set the Data Element Separation property to Tagged Encoded Length, and the actual number of data characters is less than the value found in the length field.

Field Identification

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Occurrences

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.

Message TDS properties:

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. **Note:** From Version 6.0 onwards, the use of Message Key has been deprecated for identifying an embedded message. You now have the option of identifying an embedded message by Message Identity, using the Message Alias logical property.

Simple type TDS properties:

There are no properties to show.

Unique TDS properties:

There are no properties to show.

White space characters in TDS:

White space characters are defined as ASCII characters (hexadecimal) 'X'09 to 'X'0D and EBCDIC characters 'X'05, 'X'0B, 'X'0C, 'X'0D, 'X'25, and 'X'40.

You can specify these characters in your message model using TDS mnemonics if they are important to your processing, for example, to use as group terminators or delimiting characters. See “TDS Mnemonics” on page 170 for further information.

If both the following conditions are met, white space after the end of a group and preceding the tag of the next element is ignored:

- TDS messaging standard property is "X12" or "EDIFACT"
- TDS data element separation in force for the structure is one of the following types:
  - Tagged delimiter
  - Tagged fixed length
  - Tagged encoded length

The following bit stream is accepted:

```
Tag<data>!<Any white space character>Tag
```

where:

- ! is the group terminator
- <Any white space character> is one of the ASCII or EBCDIC characters listed previously

The following X12 ASCII message successfully parses:

```
ST*856*777777%<SPC><SPC><SPC><HEX 09>BSN*00*7654321*940920*10000%
```

The sequence

```
<SPC><SPC><SPC><HEX 09>
```

is ignored by the parser.

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

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Message model object properties by object

The following objects have properties that can be viewed or set.

- "Wildcard element properties" on page 614

Attribute group reference properties

Different types of properties are available for an attribute group reference.

An attribute group reference can have the following properties;

- "Attribute group reference logical properties" on page 189
- "Attribute group reference CWF properties" on page 219
- "Attribute group reference XML properties" on page 225
- "Attribute group reference TDS properties" on page 234
- "Documentation properties for all message set objects" on page 187

Attribute group reference logical properties:

The logical properties of an attribute group reference.

Property	Type	Meaning
Reference Name	Enumerated type	The Reference Name is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Attribute group reference CWF properties:

There are no properties to show.

Attribute group reference XML properties:

The XML properties of an attribute group reference.

There are no properties to show.

Attribute group reference TDS properties:

There are no properties to show.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Attribute reference properties

Different types of properties are available for an attribute reference.

An attribute reference can have the following properties;

- "Attribute reference logical properties" on page 189
- "Attribute reference CWF properties" on page 219
- "Attribute reference XML properties" on page 225
- "Attribute reference TDS properties" on page 234

- "Documentation properties for all message set objects" on page 187

Attribute reference logical properties:

The logical properties of an attribute reference.

Property	Type	Meaning
Reference Name	Enumerated type	The Reference Name is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Property	Type	Meaning
Usage	Enumerated type	Use this property with the Value property found in an attribute object. The default value for the Usage property is optional. Select from the following options: • optional. – If the Value property is set to default, and no data has been entered in the Value property, the attribute can appear once and can have any value. – If the Value property is set to default, the attribute can appear once. If it does not appear, its value is the data that has been entered in the Value property. If it does appear, it is the value given. – Where the Value 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 Value property. If it does not appear, its value is the data that has been entered in the Value property. • prohibited. The attribute must not appear. • required. – If the Value property is set to default, and no data has been entered in the Value property, the attribute must appear once and can have any value. – If the Value property is set to fixed, the attribute must appear once, and it must match the data that has been entered in the Value property.

Attribute reference CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong

Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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	Button and Integer	If you have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1. The maximum value that you can specify is 2147483647. The default value is empty (not set).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 110.
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

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.

CWF properties for attribute reference and local attribute Boolean types:

CWF 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: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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 dateTime types:

CWF 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	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element 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. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two 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. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The dateTime is coded as a Packed Decimal number. It is valid only if the *DateTime Format* property represents numeric-only data. • `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 782 for details). • `Time Seconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *DateTime Format* property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *DateTime Format* property represents numeric-only data. The default value is `fixed length string`.
DateTime Format	String	Specify a template for date and time. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see "DateTime defaults by logical type" on page 790. See "DateTime formats" on page 782 for details of date and time formats.
Length	Button and Integer	If you have selected a *Physical Type* of `Fixed Length String`, `Packed Decimal`, or `Binary`, and have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1 for all three physical types. The maximum value that you can specify is `256` for `Fixed Length String`, `10` for `Packed Decimal`, and `2147483647` for `Binary`. The default value is 0 (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 110.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the *Physical type* property is `Packed Decimal`. By default, this check box is cleared, which indicates that the value is not signed.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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:

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, unsignedLong

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `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. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *Physical Type* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *Physical Type* 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.)
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.

Property	Type	Meaning
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a decimal element containing `1234` with a Virtual Decimal value of `3` is `1.234`, equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

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.

CWF properties for attribute reference and local attribute float types:

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 from the displayed list: • `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. • `Float`. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL and is the default value. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer`, `Packed Decimal`, and `Float` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Float`, select a value from the displayed list. The default value is 8. • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *Physical Type* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *Physical Type* 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.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select or clear (unsigned, the default) this property. If you have set *Physical Type* to `Float`, this is selected. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a float element containing `1234` with a Virtual Decimal value of `3` is `1.234`. This property is not applicable if you have set *Physical Type* to `Float`.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

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.

CWF properties for attribute reference and local attribute integer types:

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, unsignedShort

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `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. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you have set *Physical Type* to `Packed Decimal`, enter a value between 1 and 6. • If you have set *Physical Type* to `Extended Decimal`, enter a value between 1 and 11.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

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.

CWF properties for attribute reference and local attribute string types:

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, 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 from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element 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. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two 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. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. The default is Fixed Length String.
Length	Button and Integer	If you have selected a *Physical Type* of `Fixed Length String` or `Binary`, and have selected the length to be defined by *Length*, 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).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 110.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong

Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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:

XML wire format properties for attribute reference, element reference, local attribute, and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

XML properties for attribute reference, element reference, local attribute, local element Boolean types:

XML wire format properties for attribute reference, element reference, local attribute and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element dateTime types:

XML wire format properties for attribute reference, element reference, local attribute, and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

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 about the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 790. See "DateTime formats" on page 782 for details of dateTime formats.

XML properties for attribute reference, element reference, local attribute, local element decimal types:

XML wire format properties for attribute reference, element reference, local attribute, and 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, unsignedLong

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element float types:

XML wire format properties for attribute reference, element reference, local attribute, and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element integer types:

XML wire format properties for attribute reference, element reference, local attribute, and 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, unsignedShort

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element string types:

XML wire format properties for attribute reference, element reference, local attribute, and 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, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Attribute reference TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong

Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

TDS properties for attribute reference binary types:

The TDS wire format 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	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

TDS properties for attribute reference Boolean types:

There are no properties to show.

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	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

TDS properties for attribute reference decimal types:

The TDS wire format properties for attribute reference reference decimal types.

The TDS Format properties described here apply to:

- Objects: Attribute Reference
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

TDS properties for attribute reference float types:

The TDS wire format 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	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

TDS properties for attribute reference integer types:

The TDS wire format 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, unsignedShort

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

TDS properties for attribute reference string types:

The TDS wire format properties for attribute reference string types.

The TDS Format properties described here apply to:

- Objects: Attribute Reference
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Complex type properties

Different types of properties are available for a complex type.

A complex type can have the following properties;

- "Complex type logical properties" on page 190
- "Complex type CWF properties" on page 219
- "Complex type XML properties" on page 225
- "Complex type TDS properties" on page 234
- "Documentation properties for all message set objects" on page 187

Complex type logical properties:

The logical properties of a complex type include properties that describe content and substitution settings.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.
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	If this property is active, select one of the following options: • restriction. If a complex type is derived by restriction, the content model of the complex type is a subset of the base type. • 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. Derivation by list or union is not supported.

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 be edited only 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 "MRM content validation" on page 194.

Property	Type	Meaning
Local Group	Button	Select this property if the content of your complex type is a local group.
Composition	Enumerated type	Define the order, and the number of occurrences, of the elements and groups in your messages. Composition does not affect the attributes in a complex type. Select one of the following options: • Empty • sequence. If you select this option, you can define members that are elements or groups. These members, if present, must appear in the specified order in the message. They can repeat, and the same element or group can appear more than once. • choice. If you select this option, you can define members that are elements or groups. Exactly one of the defined members must be present in the message, and can repeat. 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. • all. If you select this option, you can define members that are elements; groups are not allowed. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can be used only at the top level of a complex type; it cannot be a member of another group within a type. • unorderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements can appear in any order in the message. • orderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements must appear in the specified order in the message. • message. This option is supported only by the MRM domain.If you select this option, you can define only messages as members. Each member can repeat, but the same message cannot appear twice in the list of members. Like choice, only one of the defined members can be present in a message. Unlike choice, when writing a message, if the complex type or group has more than one member, the bit stream is not padded to the length of the longest member. 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 26.

Property	Type	Meaning
Content Validation	Enumerated type	Content Validation is used only by the MRM domain. If validation is enabled in your message flow, Content Validation specifies the strictness of the MRM validation for members of a complex type or group. See “MRM content validation” on page 194 for further details. Select from the following options: • Closed. The complex type can only contain the child elements that you have added to it. • Open Defined. The complex type can contain any valid element defined within the message set. • Open. The complex type can contain any valid element, not just those that you have added to this complex type. See “Combinations of Composition and Content Validation” on page 300 for further details of these options.
Group Reference	Button	Select this option if the content of your complex type is a reference to a group object
Group Name	Enumerated type	The Group Name is the name of the group that this complex type refers to. The groups available to be referenced can be selected from the drop down list.
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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.
Mixed	Check box	Select this option when the complex type has mixed content, and contains character data and sub-elements.

Substitution settings

Property	Type	Meaning
Final	Multiple selection enumerated type	The final attribute on a complex type controls whether other types can be derived from it. Valid values are extension/restriction/all. You can select from one or more of the following: • Empty • `restriction`. Prohibit type substitution by elements whose types are restrictions of the head element's type. • `extension`. Prohibit type substitution by elements whose types are extensions of the head element's type. • `#all`. Prohibit substitution by any method. To select more than one, you will need to type the selection into the property field.

Property	Type	Meaning
Block	Multiple selection enumerated type	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: • Empty • `restriction`. Prohibit type substitution by elements whose types are restrictions of the head element's type. • `extension`. Prohibit type substitution by elements whose types are extensions of the head element's type. • `#all`. Prohibit substitution by any method. To select more than one, you will need to type the selection into the property field.
Abstract	Check box	If selected, no elements based on this type can appear in the message.

MRM content validation:

Content Validation is applied when the domain is MRM and validation is enabled. The Content Validation property specifies how strictly the MRM parser validates the members of a complex type or group.

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 30 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 30 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 cannot 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:

If your message is in the MRM domain, and validation is enabled, the members of each complex type or group are validated. The MRM validation logic is controlled by the Composition and Content Validation properties.

Content validation applies also to the IDOC domain because the IDoc parser uses the MRM parser internally. Content Validation does not affect validation in the XMLNSC domain.

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	Elements
All	Open	Elements
All	Open Defined	Elements
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:

Valid combinations of repeated and duplicate elements within a complex type depend 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:

The TDS properties of a complex type.

Field Identification

If the complex type is based on a global group, the TDS properties listed are located in the global group. If so, any changes to these properties are applied to the global group, and affect all references to the group (including any other complex types which are based on it).

Property	Type	Meaning
Data Element Separation	Enumerated type	Select one of the following values to specify the method that is used to separate the data elements within the type. • Tagged Delimited. This value indicates that all elements within the complex type are identified by a tag, and, if a value is specified in the optional Delimiter property, are separated by that value. You must set the Tag property for all child elements of simple type, and you can set the Delimiter property to a non-empty value. See "Global element TDS properties" on page 240. You must also set the Tag Data Separator or the Length of Tag property. • 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 Tag property for each of the child elements of this complex type, and each child element must have a Length or Length Reference property assigned to it. You must also set the Tag Data Separator or the Length of Tag property. • 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 the Length of Tag property, or variable length delimited by the Tag Data Separator property. You must also set the Length Of Encoded Length property so that the parser knows the size of the length field, and set the Extra Chars in Encoded Length property. This property tells the parser what to subtract from the value in the Length Of Encoded Length property to get the actual length of the data that follows the length field. This method provides a more flexible way of handling ACORD AL3 standard messages than using the Fixed Length AL3 value, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard. • All Elements Delimited. This value indicates that all elements within the complex type are separated by a delimiter. You must set a value in the Delimiter property. • Variable Length Elements Delimited. This value indicates that some of the elements within the complex type might be of variable length. Variable length elements must be delimited by the value specified in the Delimiter property. • Use Data Pattern. This value indicates that the parser determines the elements by matching the data with the regular expression that is set in the Data Pattern property of the element or type member. See "Message definition file properties" on page 185. • 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 Length property to the offset. See "Global element TDS properties" on page 240. If you set the Data Element Separation property of a complex type to Fixed Length, you must also set the Data Element Separation property of all complex children of this type to Fixed Length. Each child element must have a Length or Length Reference property assigned to it. • 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 regarding missing optional elements, encoded lengths, and versioning, must be applied. If you set the Data Element Separation property of a complex type to Fixed Length AL3, you must also set the Data Element Separation property of all complex children of this type to Fixed Length AL3. • Undefined. This value is set automatically if you set the Type Composition property of a complex type to Message, and you cannot change it to any other value. Do not set the Type Composition property to Empty, Choice, Unordered Set, Ordered Set, Sequence, or Simple Unordered Set. If you do, you cannot check in the type.

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character, or string, that precedes the data that belongs to a group, or a complex type, within the bit stream.
Group Terminator	String	Specify the value of a special character, or string, that terminates the data that belongs 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 that is used between data elements. This property applies only to the delimited Data Element Separation methods (Tagged Delimited, All Elements Delimited, and Variable Length Elements Delimited).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select whether you want delimiters to be suppressed for elements that are missing within a message. Select from: • 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 three elements and only two are present, the last delimiter can be omitted from the message. • Never. Use this option to ensure that even if optional elements are not present, all delimiters are written out. Use this option when the same delimiter is used to delimit parent and child objects. For example, if an optional child element is missing, message processing applications cannot tell where the child elements in a message end and the next parent element starts, if the delimiters are all the same.
Observe Element Length	Check box	This property is applicable when Data Element Separation is All Elements Delimited or Tagged Delimited. Select this check box if the Length property of child simple elements is significant when parsing and writing. • During parsing, an exception is thrown if the length of the extracted data exceeds the specified length. Otherwise, the data is trimmed according to the Justification and Padding Character properties of the child element. • During writing, an exception is thrown if the data to write exceeds the specified length. Otherwise, the data is padded according to the Justification and Padding Character properties of the child element. Clear this check box to ignore the Length property when parsing and writing. The default value depends on the setting of the Messaging Standard property (at the message set level) and the Data Element Separation property. • If Data Element Separation is All Elements Delimited and the Messaging Standard is TLOG, the check box is selected. • If Data Element Separation is All Elements Delimited and the Messaging Standard is other than TLOG, the check box is cleared. • If Data Element Separation is Tagged Delimited, the check box is cleared. For all other data element separation methods, the check box is disabled and does not influence the behavior of the TDS parser.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The Tag Data Separator and Length of Tag properties are mutually exclusive. If you set the property Tag Data Separator, it overrides Length of Tag. This property applies only to the tagged Data Element Separation methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).

Property	Type	Meaning
Length of Tag	Button and Integer	Specify the length of a tag value. When the message is parsed, this property allows tags to be extracted from the bit stream if the Tag Data Separator property is not set. The Tag Data Separator and Length of Tag properties are mutually exclusive. If you set the property Tag Data Separator, it overrides this value. This property applies only to the tagged Data Element Separation methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).
Length of Encoded Length	Integer	Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647. You must set this property if you have set the Data Element Separation property to Tagged Encoded Length; it is not valid otherwise. The actual number of data characters that are parsed depends on the value of the Extra Chars in Encoded Length property.
Extra Chars in Encoded Length	Integer	(Only valid if the Data Element Separation 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.) 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. You must set this property if you have set the Data Element Separation property to Tagged Encoded Length, and the actual number of data characters is less than the value found in the length field.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Element reference properties

Different types of properties are available for an element reference.

An element reference can have the following properties:

- "Element reference logical properties" on page 195
- "Element reference CWF properties" on page 219
- "Element reference XML properties" on page 225
- "Element reference TDS properties" on page 238
- "Documentation properties for all message set objects" on page 187

Element reference logical properties:

The logical properties of an element reference include properties that specify the number of occurrences of the element reference.

Property	Type	Meaning
Reference Name	Enumerated type	The Reference Name is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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 element's Value properties. The following table 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 Value property. In this example, the element must contain the text Delta.
2	-1	Delta		The element must appear twice or more, and it must match the data that has been entered in the Value property. In this example, at least two elements must contain the text Delta.
0	1			The element is optional, can appear once, and can 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 Value property. If it does not appear, its value is the data that has been entered in the Value property.
0	1		Delta	The element is optional, and can appear once. If it does not appear, its value is the data that has been entered in the Value property. If it does appear, it must be the value given in the element.

Min Occurs	Max Occurs	Fixed	Default	Notes
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 Value property, else it is the value given in the element.
0	0			The element is prohibited, and must not appear.

Element reference CWF properties:

The properties, and permissible values, vary according to the type of object.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong
Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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:

CWF wire format 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	Button and Integer	If you have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1. The maximum value that you can specify is 2147483647. The default value is empty (not set).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 110.
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* 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 CWF wire format 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: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* 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 CWF wire format 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	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element 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. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two 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. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The dateTime is coded as a Packed Decimal number. It is valid only if the *DateTime Format* property represents numeric-only data. • `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 782 for details). • `Time Seconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *DateTime Format* property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *DateTime Format* property represents numeric-only data. The default value is `fixed length string`.
Length	Button and Integer	If you have selected a *Physical Type* of `Fixed Length String`, `Packed Decimal`, or `Binary`, and have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1 for all three physical types. The maximum value that you can specify is `256` for `Fixed Length String`, `10` for `Packed Decimal`, and `2147483647` for `Binary`. The default value is 0 (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 110.
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.
DateTime Format	String	Specify a template for date and time. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 790. See “DateTime formats” on page 782 for details of date and time formats.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the *Physical type* property is `Packed Decimal`. By default, this check box is cleared, which indicates that the value is not signed.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list. The option that you select determines the value that you must set for the property *Encoding Null Value*: • `NULLPadFill`. This option is valid only if *Physical Type* is `Fixed Length String`. The field is filled with the value specified by the *Padding Character*. The default value. • `NULLLogicalValue`. The *Encoding Null Value* property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This specifies that *Encoding Null Value* contains a value that is directly substituted as if it is a string. Use this option when the value you have set for *Encoding Null Value* to specify a null date is not a dateTime value, or does not conform to the standard dateTime format `yyyy-MM-dd 'T'HH:mm:ss`. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.
Encoding Null Value	String	If you set the *Encoding Null* property to `NULLPadFill`, this property is disabled. If you set the *Encoding Null* property to `NULLLogicalValue`, you must set this property to an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value conforming to `yyyy-MM-dd'T'HH:mm:ss` such as `1970-12-01`. If you set the *Encoding Null* property to `NULLLiteralValue`, you can enter any value that is the same length as the field. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* 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 CWF wire format 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, unsignedLong

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `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. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *Physical Type* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *Physical Type* 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.)
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.

Property	Type	Meaning
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a decimal element containing `1234` with a Virtual Decimal value of `3` is `1.234`, equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This option is valid only if *Physical Type* is `External Decimal`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for `NULLLiteralFill`. The default value is zero. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed list • Enter a character between quotation marks, for example `'c'` or `"c"`, where `c` is any alphanumeric character. • Enter a hexadecimal character 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for element reference and local element float types:

The CWF wire format 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	Select one from the displayed list: • `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. • `Float`. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL and is the default value. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer`, `Packed Decimal`, and `Float` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Float`, select a value from the displayed list. The default value is 8. • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *Physical Type* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *Physical Type* 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.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select or clear (unsigned, the default) this property. If you have set *Physical Type* to `Float`, this is selected. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a float element containing `1234` with a Virtual Decimal value of `3` is `1.234`. This property is not applicable if you have set *Physical Type* to `Float`.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This option is valid only if *Physical Type* is `External Decimal`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for `NULLLiteralFill`. The default value is zero. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed list • Enter a character between quotation marks, for example `'c'` or `"c"`, where `c` is any alphanumeric character. • Enter a hexadecimal character 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for element reference and local element integer types:

The CWF wire format 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, unsignedShort

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `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. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you have set *Physical Type* to `Packed Decimal`, enter a value between 1 and 6. • If you have set *Physical Type* to `Extended Decimal`, enter a value between 1 and 11.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This option is valid only if *Physical Type* is `External Decimal`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for `NULLLiteralFill`. The default value is zero. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed list • Enter a character between quotation marks, for example `'c'` or `"c"`, where `c` is any alphanumeric character. • Enter a hexadecimal character 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for element reference and local element string types:

The CWF wire format 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, 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 from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element 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. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two 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. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. The default is Fixed Length String.
Length	Button and Integer	If you have selected a *Physical Type* of `Fixed Length String` or `Binary`, and have selected the length to be defined by *Length*, 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).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 110.
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This is valid only if *Physical Type* is `Fixed Length String`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.
Encoding Null Value	STRING	The use of this property depends on the *Encoding Null* property. If specified, its length must be equal to the length of the string element, except for `NULLLiteralFill`. The default value is empty (not set). If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Element reference XML properties:

The properties, and their permissible values, vary according to the type of object.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong
Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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:

XML wire format properties for attribute reference, element reference, local attribute, and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

XML properties for attribute reference, element reference, local attribute, local element Boolean types:

XML wire format properties for attribute reference, element reference, local attribute and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element dateTime types:

XML wire format properties for attribute reference, element reference, local attribute, and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

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 about the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 790. See "DateTime formats" on page 782 for details of dateTime formats.

XML properties for attribute reference, element reference, local attribute, local element decimal types:

XML wire format properties for attribute reference, element reference, local attribute, and 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, unsignedLong

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element float types:

XML wire format properties for attribute reference, element reference, local attribute, and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element integer types:

XML wire format properties for attribute reference, element reference, local attribute, and 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, unsignedShort

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element string types:

XML wire format properties for attribute reference, element reference, local attribute, and 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, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Element reference TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types - base64Binary - hexBinary	Boolean types - Boolean	Complex types
DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong	Float types - double - float

Integer types	Interval types	String types
- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

TDS properties for element reference binary types:

TDS wire format properties for element reference binary types.

The TDS Format properties described here apply to:

- Objects: Element Reference
- Binary schema types: base64Binary, hexBinary

Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

TDS properties for element reference Boolean types:

TDS wire format properties for element reference Boolean types.

The TDS Format properties described here apply to:

- Objects: Element Reference
- Boolean schema types: Boolean

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

TDS properties for element references to complex elements:

The TDS Format properties that apply to element references where the global element is of complex type.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

TDS properties for element reference dateTime types:

TDS wire format properties for attribute reference and 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

Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

TDS properties for element reference decimal types:

The TDS wire format properties for attribute reference and element reference decimal types.

The TDS Format properties described here apply to:

- Objects: Element Reference

- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

TDS properties for element reference float types:

TDS wire format properties for attribute reference and element reference float types.

The TDS Format properties described here apply to:

- Objects: Element Reference
- Float schema types: double, float

Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

TDS properties for element reference integer types:

TDS wire format properties for attribute reference and element reference integer types.

The TDS Format properties described here apply to:

- Objects: Element Reference
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

TDS properties for element reference string types:

The TDS wire format properties for attribute reference and element reference string types.

The TDS Format properties described here apply to:

- Objects: Element Reference
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Physical representation

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Global attribute properties

Different types of properties are available for a global attribute.

A global attribute can have the following properties;

- "Global attribute logical properties" on page 196
- "Global attribute CWF properties" on page 220
- "Global attribute XML properties" on page 226
- "Global attribute TDS properties" on page 239
- "Documentation properties for all message set objects" on page 187

Global attribute logical properties:

The logical properties of a global attribute.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • `-` the hyphen • `_` the underscore • `.` the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • `int` • `string` • `Boolean` • `hexBinary` • `dateTime` • `date` • `time` • `decimal` • `float` • `(More...)` • `(New Simple Type)` • `(New Complex Type)` If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.

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 are displayed in the drop-down list.

Value

The *Value* properties are used with the *Usage* property in an Attribute Reference or a Local Attribute.

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.

Property	Type	Meaning
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.
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. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds 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 remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Global attribute CWF properties:

There are no properties to show.

Global attribute XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong
Float types	**Integer types**	**Interval types**	**String types**
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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 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	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 (_).

Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

XML properties for global attribute and global element Boolean types:

The XML wire format 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	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 (_).

XML properties for global attribute and global element dateTime types:

The XML wire format 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	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 (_).

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 about the defaults for the dateTime format according to the logical type see “DateTime defaults by logical type” on page 790. See “DateTime formats” on page 782 for details of dateTime formats.

XML properties for global attribute and global element decimal types:

The XML wire format 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, unsignedLong

Property	Type	Meaning
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 (_).

XML properties for global attribute and global element float types:

The XML wire format 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	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 (_).

XML properties for global attribute and global element integer types:

The XML wire format 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, unsignedShort

Property	Type	Meaning
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 (_).

XML properties for global attribute and global element string types:

The XML wire format 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, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Property	Type	Meaning
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 (_).

Global attribute TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong
Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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 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	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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.

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	This value of this property defaults to Binary. It cannot be changed.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. The default is dependent on the setting of the message set property Derive default length from logical type. If Derive default length from logical type is selected, the default value is derived from any length or maxLength value constraint (schema facet) on the object's simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.

TDS properties for global attribute Boolean types:

The TDS format 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	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Binary. The data is in bit string format. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.

TDS properties for global attribute dateTime types:

The TDS format 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	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Length Encoded String 1. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a Length Encoded String 1 element is 255 length units. • Length Encoded String 2. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a Length Encoded String 2 element is 65535 length units. • Null Terminated String. The data string ends with the hexadecimal NULL character, `X'00'`. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • Binary. The data is in bit string format. • Time Seconds. This value supports C time_t, and Java Date and Time objects. It is valid only if the DateTime Format property represents numeric-only data. • Time Milliseconds. This value supports C time_t, and Java Date and Time objects. It is valid only if the DateTime Format property represents numeric-only data. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
DateTime Format	String	Specify a template for date and time. The default DateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 790. See “DateTime formats” on page 782 for details of date and time formats.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the Physical type property is Packed Decimal. By default, this check box is cleared, which indicates that the value is not signed.

TDS properties for global attribute decimal types:

The TDS format 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, unsignedLong

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see "Regular expression syntax" on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • TLOG Specific - this option can be selected only if the Message Standard property of the message set is TLOG. This option indicates that the format of the data is specific to the TLOG messaging standard. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the Precision property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if Physical Type is Text. It is used if the value of the Virtual Decimal Point property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set Precision to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set Precision 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 Precision to 2. • If you set Precision 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 Precision to 5. Or select one option from the list: • All Significant Digits - decimal separator only required if fractional digits (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • Explicit Decimal Separator - decimal separator always required: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present.

TDS properties for global attribute float types:

The TDS format 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	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Float. Equates to the data type FLOAT or DOUBLE in C, or the COMP-1 or COMP-2 numeric data type in COBOL. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the Precision property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if Physical Type is Text. It is used if the value of the Virtual Decimal Point property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set Precision to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set Precision 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 Precision to 2. • If you set Precision 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 Precision to 5. Or select one option from the list: • All Significant Digits - decimal separator only required if fractional digits (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • Explicit Decimal Separator - decimal separator always required: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present. • Exponential Notation - Example "1.23456e002": data is written out to the bit stream as a signed value having the format `[sign1]a.bbbe[sign2]ccc` where: – `[sign1]` is the value of Negative Sign if the value is negative – `a` is a single decimal digit – `bbb` is one or more decimal digits – `[sign2]` is the value of Negative Sign if the exponent is negative – `ccc` is exactly three decimal digits (the exponent) `[sign1]` and `[sign2]` are absent if the value and exponent are positive. 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 that the value of Negative Sign is "-", and the value of Sign Orientation is `Leading`. The value -0.00012 is represented as `1.2*e*004` if Negative Sign is "*" and Sign Orientation is Trailing.

TDS properties for global attribute integer types:

The TDS format 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, unsignedShort

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

TDS properties for global attribute string types:

The TDS format properties for global attribute string types.

The TDS Format properties described here apply to:

- Objects: Global Attribute
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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.

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.
Interpret Element Value	Enumerated type	Specify whether values stored within this object are interpreted as having significance for the parser and, if so, the type of interpretation that occurs. This interpretation is standard-specific and is therefore hard coded. The possible values for this property are: • None (the default value) • EDIFACT Service String • X12 Service String • Message Key • EDIFACT Syntax Level ID • HL7 Service String • HL7 Field Separator **Note:** The Message Key enumeration has been deprecated.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Length Encoded String 1. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a Length Encoded String 1 element is 255 length units. • Length Encoded String 2. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a Length Encoded String 2 element is 65535 length units. • Null Terminated String. The data string ends with the hexadecimal NULL character, `X'00'`. • TLOG Specific - this option can be selected only if the Message Standard property of the message set is TLOG. This option indicates that the format of the data is specific to the TLOG messaging standard. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Global attribute group properties

Different types of properties are available for a global attribute group.

A global attribute group can have the following properties;

- "Global attribute group logical properties" on page 199
- "Global attribute group CWF properties" on page 220
- "Global attribute group XML properties" on page 227
- "Global attribute group TDS properties" on page 240
- "Documentation properties for all message set objects" on page 187

Global attribute group logical properties:

The logical properties of an attribute group.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.

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:

The TDS properties of a global attribute group.

There are no properties to show.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Global element properties

Different types of properties are available for a global element.

A global element can have the following properties;

- "Global element logical properties" on page 200
- "Global element CWF properties" on page 220
- "Global element XML properties" on page 227
- "Global element TDS properties" on page 240
- "Documentation properties for all message set objects" on page 187

Global element logical properties:

The logical properties of a global element.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.

Property	Type	Meaning
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • int • string • Boolean • hexBinary • dateTime • date • time • decimal • float • (More...) • (New Simple Type) • (New Complex Type) If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.
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 are displayed in the drop-down list.

Value

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.
Nillable	Check box	Select this option if you want the element to be able to be defined as null. A null value is distinct from being empty, when the element contains no data.

Property	Type	Meaning
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. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds 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 remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

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	Limit the set of elements that can belong to its substitution group. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element. • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element. • #all. Prohibit substitution by all methods.
Block	Enumerated type	Limit the set of elements that can be substituted for this element in a message. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element • substitution. Prohibit element substitution by members of the element's substitution group. • #all. Prohibit substitution by all methods.
Substitution Group	Enumerated type	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 option 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 CWF properties to show for a global element.

Global element XML properties:

The properties, and their permissible values, vary according to the type of object.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong
Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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 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	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 (_).

Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

XML properties for global attribute and global element Boolean types:

The XML wire format 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	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 (_).

XML properties for global attribute and global element dateTime types:

The XML wire format 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	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 (_).

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 about the defaults for the dateTime format according to the logical type see “DateTime defaults by logical type” on page 790. See “DateTime formats” on page 782 for details of dateTime formats.

XML properties for global attribute and global element decimal types:

The XML wire format 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, unsignedLong

Property	Type	Meaning
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 (_).

XML properties for global attribute and global element float types:

The XML wire format 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	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 (_).

XML properties for global attribute and global element integer types:

The XML wire format 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, unsignedShort

Property	Type	Meaning
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 (_).

XML properties for global attribute and global element string types:

The XML wire format 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, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Property	Type	Meaning
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 (_).

Global element TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	Complex types
- base64Binary - hexBinary	- Boolean	
DateTime types	Decimal types	Float types
- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong	- double - float
Integer types	Interval types	String types
- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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 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	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	For all Messaging Standard values, the Physical Type property is set to Binary and cannot be changed.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. The default is dependent on the setting of the message set property Derive default length from logical type. If Derive default length from logical type is selected, the default value is derived from any length or maxLength value constraint (schema facet) on the object's simple type.
Length Units	Enumerated type	Always set to Bytes.

TDS properties for global element Boolean types:

The TDS format 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	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see "Regular expression syntax" on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Binary. The data is in bit string format. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.

TDS properties for global element of complex type:

The TDS Format properties that apply to global elements of complex type.

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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.

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

TDS properties for global element dateTime types:

The TDS format 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	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Length Encoded String 1. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a Length Encoded String 1 element is 255 length units. • Length Encoded String 2. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a Length Encoded String 2 element is 65535 length units. • Null Terminated String. The data string ends with the hexadecimal NULL character, `X'00'`. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • Binary. The data is in bit string format. • Time Seconds. This value supports C time_t, and Java Date and Time objects. It is valid only if the DateTime Format property represents numeric-only data. • Time Milliseconds. This value supports C time_t, and Java Date and Time objects. It is valid only if the DateTime Format property represents numeric-only data. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if one of the following statements is true: • Physical Type is Packed Decimal. • Physical Type is Text, no Length Reference is specified, and the Data Element Separation of the parent complex type or group is Fixed Length, Tagged Fixed Length, or Fixed Length AL3. The default is dependent on the physical type of the object. If Physical Type is Length Encoded String 1, Length Encoded String 2, or Null Terminated String, this property is not applicable. If Physical Type is Time Seconds, the value of this property is 4, and cannot be changed. If Physical Type is Time Milliseconds, the value of this property is 8, and cannot be changed.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
DateTime Format	String	Specify a template for date and time. The default DateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 790. See “DateTime formats” on page 782 for details of date and time formats.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the Physical type property is Packed Decimal. By default, this check box is cleared, which indicates that the value is not signed.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see "TDS Null handling options" on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

TDS properties for global element decimal types:

The TDS format 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, unsignedLong

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • TLOG Specific - this option can be selected only if the Message Standard property of the message set is TLOG. This option indicates that the format of the data is specific to the TLOG messaging standard. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any minInclusive, maxInclusive, minExclusive, or maxExclusive value constraints (schema facets), on the simple type.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed.
Sign EBCDIC Custom Overpunched	Check box	Specify whether EBCDIC custom sign format is used. This property is applicable only if the Signed property is selected and the Physical Type property is set to External Decimal.

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the Precision property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if Physical Type is Text. It is used if the value of the Virtual Decimal Point property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set Precision to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set Precision 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 Precision to 2. • If you set Precision 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 Precision to 5. Or select one option from the list: • All Significant Digits - decimal separator only required if fractional digits (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • Explicit Decimal Separator - decimal separator always required: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 770.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

TDS properties for global element float types:

The TDS format 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	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see "Regular expression syntax" on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Float. Equates to the data type FLOAT or DOUBLE in C, or the COMP-1 or COMP-2 numeric data type in COBOL. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed.
Sign EBCDIC Custom Overpunched	Check box	Specify whether EBCDIC custom sign format is used. This property is applicable only if the Signed property is selected and the Physical Type property is set to External Decimal.

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the Precision property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if Physical Type is Text. It is used if the value of the Virtual Decimal Point property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set Precision to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set Precision 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 Precision to 2. • If you set Precision 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 Precision to 5. Or select one option from the list: • All Significant Digits - decimal separator only required if fractional digits (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • Explicit Decimal Separator - decimal separator always required: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present. • Exponential Notation - Example "1.23456e002": data is written out to the bit stream as a signed value having the format `[sign1]a.bbbe[sign2]ccc` where: – `[sign1]` is the value of Negative Sign if the value is negative – `a` is a single decimal digit – `bbb` is one or more decimal digits – `[sign2]` is the value of Negative Sign if the exponent is negative – `ccc` is exactly three decimal digits (the exponent) `[sign1]` and `[sign2]` are absent if the value and exponent are positive. 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 that the value of Negative Sign is "-", and the value of Sign Orientation is `Leading`. The value -0.00012 is represented as `1.2*e*004` if Negative Sign is "*" and Sign Orientation is Trailing.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see "TDS Null handling options" on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

TDS properties for global element integer types:

The TDS format 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, unsignedShort

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any minInclusive, maxInclusive, minExclusive, or maxExclusive value constraints (schema facets), on the simple type.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed.
Sign EBCDIC Custom Overpunched	Check box	Specify whether EBCDIC custom sign format is used. This property is applicable only if the Signed property is selected and the Physical Type property is set to External Decimal.

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 770.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

TDS properties for global element interval types:

The TDS format properties for global element string types.

The TDS Format properties described here apply to:

- Objects: Global Element
- Interval schema types: duration

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see "Regular expression syntax" on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Length Encoded String 1. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a Length Encoded String 1 element is 255 length units. • Length Encoded String 2. The data's first 2 bytes contains the length (in length units) of the data string that follows the two length bytes. The maximum length of a Length Encoded String 2 element is 65535 length units. • Null Terminated String. The data string ends with the hexadecimal NULL character, `X'00'`. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see "TDS Null handling options" on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

TDS properties for global element string types:

The TDS format properties for global element string types.

The TDS Format properties described here apply to:

- Objects: Global Element
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.
Interpret Element Value	Enumerated type	Specify whether values stored within this object are interpreted as having significance for the parser and, if so, the type of interpretation that occurs. This interpretation is standard-specific and is therefore hard coded. The possible values for this property are: • None (the default value) • EDIFACT Service String • X12 Service String • Message Key • EDIFACT Syntax Level ID • HL7 Service String • HL7 Field Separator **Note:** The Message Key enumeration has been deprecated.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Length Encoded String 1. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a Length Encoded String 1 element is 255 length units. • Length Encoded String 2. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a Length Encoded String 2 element is 65535 length units. • Null Terminated String. The data string ends with the hexadecimal NULL character, `X'00'`. • TLOG Specific - this option can be selected only if the Message Standard property of the message set is TLOG. This option indicates that the format of the data is specific to the TLOG messaging standard. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.

Property	Type	Meaning
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see "TDS Null handling options" on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Global group properties

Different types of properties are available for a global group.

A global element can have the following properties;

- "Global group logical properties" on page 203
- "Global group CWF properties" on page 220
- "Global group XML properties" on page 228
- "Global group TDS properties" on page 241
- "Documentation properties for all message set objects" on page 187

Global group logical properties:

The logical properties of a global group.

Valid children in a global group that depend on both **Composition** and **Content Validation** are shown in "MRM content validation" on page 194.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.

Property	Type	Meaning
Composition	Enumerated type	Define the order, and the number of occurrences, of the elements and groups in your messages. Composition does not affect the attributes in a complex type. Select from the following options: • Empty • sequence. If you select this option, you can define members that are elements or groups. These members, if present, must appear in the specified order in the message. They can repeat, and the same element or group can appear more than once. • choice. If you select this option, you can define members that are elements or groups. Exactly one of the defined members must be present in the message, and can repeat. 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. • all. If you select this option, you can define members that are elements; groups are not allowed. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can be used only at the top level of a complex type; it cannot be a member of another group within a type. • unorderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements can appear in any order in the message. • orderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements must appear in the specified order in the message. • message. This option is supported only by the MRM domain. If you select this option, you can define only messages as members. Each member can repeat, but the same message cannot appear twice in the list of members. Like choice, only one of the defined members can be present in a message. Unlike choice, when writing a message, if the complex type or group has more than one member, the bit stream is not padded to the length of the longest member. 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 26.

Property	Type	Meaning
Content Validation	Enumerated type	Content Validation is used only by the MRM domain. If validation is enabled in your message flow, Content Validation specifies the strictness of the MRM validation for members of a complex type or group. See “MRM content validation” on page 194 for further details. Select from the following options: • Closed. The complex type can only contain the child elements that you have added to it. • Open Defined. The complex type can contain any valid element defined within the message set. • Open. The complex type can contain any valid element, not just those that you have added to this complex type. See “Combinations of Composition and Content Validation” on page 300 for further details of these options.

Global group CWF properties:

There are no properties to show.

Global group XML properties:

There are no properties to show.

There are no properties to show.

Global group TDS properties:

The TDS properties of a global group.

Field Identification

Property	Type	Meaning
Data Element Separation	Enumerated type	Select one of the following values to specify the method that is used to separate the data elements within the type. • Tagged Delimited. This value indicates that all elements within the complex type are identified by a tag, and, if a value is specified in the optional Delimiter property, are separated by that value. You must set the Tag property for all child elements of simple type, and you can set the Delimiter property to a non-empty value. See "Global element TDS properties" on page 240. You must also set the Tag Data Separator or the Length of Tag property. • 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 Tag property for each of the child elements of this complex type, and each child element must have a Length or Length Reference property assigned to it. You must also set the Tag Data Separator or the Length of Tag property. • 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 the Length of Tag property, or variable length delimited by the Tag Data Separator property. You must also set the Length Of Encoded Length property so that the parser knows the size of the length field, and set the Extra Chars in Encoded Length property. This property tells the parser what to subtract from the value in the Length Of Encoded Length property to get the actual length of the data that follows the length field. This method provides a more flexible way of handling ACORD AL3 standard messages than using the Fixed Length AL3 value, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard. • All Elements Delimited. This value indicates that all elements within the complex type are separated by a delimiter. You must set a value in the Delimiter property. • Variable Length Elements Delimited. This value indicates that some of the elements within the complex type might be of variable length. Variable length elements must be delimited by the value specified in the Delimiter property. • Use Data Pattern. This value indicates that the parser determines the elements by matching the data with the regular expression that is set in the Data Pattern property of the element or type member. See "Message definition file properties" on page 185. • 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 Length property to the offset. See "Global element TDS properties" on page 240. If you set the Data Element Separation property of a complex type to Fixed Length, you must also set the Data Element Separation property of all complex children of this type to Fixed Length. Each child element must have a Length or Length Reference property assigned to it. • 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 regarding missing optional elements, encoded lengths, and versioning, must be applied. If you set the Data Element Separation property of a complex type to Fixed Length AL3, you must also set the Data Element Separation property of all complex children of this type to Fixed Length AL3. • Undefined. This value is set automatically if you set the Type Composition property of a complex type to Message, and you cannot change it to any other value. Do not set the Type Composition property to Empty, Choice, Unordered Set, Ordered Set, Sequence, or Simple Unordered Set. If you do, you cannot check in the type.

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character, or string, that precedes the data that belongs to a group, or a complex type, within the bit stream.
Group Terminator	String	Specify the value of a special character, or string, that terminates the data that belongs 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 that is used between data elements. This property applies only to the delimited Data Element Separation methods (Tagged Delimited, All Elements Delimited, and Variable Length Elements Delimited).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select whether you want delimiters to be suppressed for elements that are missing within a message. Select from: • 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 three elements and only two are present, the last delimiter can be omitted from the message. • Never. Use this option to ensure that even if optional elements are not present, all delimiters are written out. Use this option when the same delimiter is used to delimit parent and child objects. For example, if an optional child element is missing, message processing applications cannot tell where the child elements in a message end and the next parent element starts, if the delimiters are all the same.
Observe Element Length	Check box	This property is applicable when Data Element Separation is All Elements Delimited or Tagged Delimited. Select this check box if the Length property of child simple elements is significant when parsing and writing. • During parsing, an exception is thrown if the length of the extracted data exceeds the specified length. Otherwise, the data is trimmed according to the Justification and Padding Character properties of the child element. • During writing, an exception is thrown if the data to write exceeds the specified length. Otherwise, the data is padded according to the Justification and Padding Character properties of the child element. Clear this check box to ignore the Length property when parsing and writing. The default value depends on the setting of the Messaging Standard property (at the message set level) and the Data Element Separation property. • If Data Element Separation is All Elements Delimited and the Messaging Standard is TLOG, the check box is selected. • If Data Element Separation is All Elements Delimited and the Messaging Standard is other than TLOG, the check box is cleared. • If Data Element Separation is Tagged Delimited, the check box is cleared. For all other data element separation methods, the check box is disabled and does not influence the behavior of the TDS parser.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The Tag Data Separator and Length of Tag properties are mutually exclusive. If you set the property Tag Data Separator, it overrides Length of Tag. This property applies only to the tagged Data Element Separation methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).

Property	Type	Meaning
Length of Tag	Button and Integer	Specify the length of a tag value. When the message is parsed, this property allows tags to be extracted from the bit stream if the Tag Data Separator property is not set. The Tag Data Separator and Length of Tag properties are mutually exclusive. If you set the property Tag Data Separator, it overrides this value. This property applies only to the tagged Data Element Separation methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).
Length of Encoded Length	Integer	Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647. You must set this property if you have set the Data Element Separation property to Tagged Encoded Length; it is not valid otherwise. The actual number of data characters that are parsed depends on the value of the Extra Chars in Encoded Length property.
Extra Chars in Encoded Length	Integer	(Only valid if the Data Element Separation 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.) 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. You must set this property if you have set the Data Element Separation property to Tagged Encoded Length, and the actual number of data characters is less than the value found in the length field.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Group reference properties

Different types of properties are available for a group reference.

A group reference can have the following properties;

- "Group reference logical properties" on page 205
- "Group reference CWF properties" on page 220
- "Group reference XML properties" on page 228
- "Group reference TDS properties" on page 244
- "Documentation properties for all message set objects" on page 187

Group reference logical properties:

The logical properties of a group reference include properties that specify the number of occurrences of the group reference.

Property	Type	Meaning
Reference Name	Enumerated type	The Reference Name is the name of the object that this object is referring to. The objects available to reference can be selected from the list.

Occurrence properties

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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.

Group reference CWF properties:

The CWF properties of a group reference.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Group reference XML properties:

The XML properties of a group reference.

There are no properties to show.

Group reference TDS properties:

The following tables describe the TDS properties of a group reference.

Field identification

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Occurrences

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.
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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Key properties

The different types of properties available for a key.

A key can have the following properties;

- "Key logical properties" on page 205
- "Key CWF properties" on page 221
- "Key XML properties" on page 228
- "Key TDS properties" on page 245
- "Documentation properties for all message set objects" on page 187

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:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Keyref properties

The different types of properties available for a keyref.

A keyref can have the following properties;

- "Keyref logical properties" on page 205
- "Keyref CWF properties" on page 221
- "Keyref XML properties" on page 228
- "Keyref TDS properties" on page 245
- "Documentation properties for all message set objects" on page 187

Keyref logical properties:

This describes the logical properties of a keyref.

There are no properties to show.

Keyref CWF properties:

This describes the CWF properties of a keyref.

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:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Local attribute properties

Different types of properties are available for a local attribute.

A local attribute can have the following properties;

- "Local attribute logical properties" on page 206
- "Local attribute CWF properties" on page 221
- "Local attribute XML properties" on page 228
- "Local attribute TDS properties" on page 245
- "Documentation properties for all message set objects" on page 187

Local attribute logical properties:

The logical properties of a local attribute.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.

Property	Type	Meaning
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • int • string • Boolean • hexBinary • dateTime • date • time • decimal • float • (More...) • (New Simple Type) • (New Complex Type) If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.
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 are displayed in the drop-down list.

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	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.

Property	Type	Meaning
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. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds 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 remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Usage properties

Property	Type	Meaning
Usage	Enumerated type	Use this property with the Value property found in an attribute object. The default value for the Usage property is optional. Select from the following options: • optional. – If the Value property is set to default, and no data has been entered in the Value property, the attribute can appear once and can have any value. – If the Value property is set to default, the attribute can appear once. If it does not appear, its value is the data that has been entered in the Value property. If it does appear, it is the value given. – Where the Value 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 Value property. If it does not appear, its value is the data that has been entered in the Value property. • prohibited. The attribute must not appear. • required. – If the Value property is set to default, and no data has been entered in the Value property, the attribute must appear once and can have any value. – If the Value property is set to fixed, the attribute must appear once, and it must match the data that has been entered in the Value property.

Local attribute CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong
Float types	**Integer types**	**Interval types**	**String types**
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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	Button and Integer	If you have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1. The maximum value that you can specify is `2147483647`. The default value is empty (not set).

Property	Type	Meaning
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 110.
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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 Boolean types:

CWF 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: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

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.

CWF properties for attribute reference and local attribute dateTime types:

CWF 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	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element 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. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two 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. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The dateTime is coded as a Packed Decimal number. It is valid only if the *DateTime Format* property represents numeric-only data. • `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 782 for details). • `Time Seconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *DateTime Format* property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *DateTime Format* property represents numeric-only data. The default value is `fixed length string`.
DateTime Format	String	Specify a template for date and time. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see "DateTime defaults by logical type" on page 790. See "DateTime formats" on page 782 for details of date and time formats.

Property	Type	Meaning
Length	Button and Integer	If you have selected a *Physical Type* of `Fixed Length String`, `Packed Decimal`, or `Binary`, and have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is `1` for all three physical types. The maximum value that you can specify is `256` for `Fixed Length String`, `10` for `Packed Decimal`, and `2147483647` for `Binary`. The default value is 0 (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 110.
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the *Physical type* property is `Packed Decimal`. By default, this check box is cleared, which indicates that the value is not signed.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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:

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, unsignedLong

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `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. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *Physical Type* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *Physical Type* 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.)
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.

Property	Type	Meaning
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a decimal element containing `1234` with a Virtual Decimal value of `3` is `1.234`, equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

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.

CWF properties for attribute reference and local attribute float types:

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 from the displayed list: • `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. • `Float`. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL and is the default value. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer`, `Packed Decimal`, and `Float` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Float`, select a value from the displayed list. The default value is 8. • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *Physical Type* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *Physical Type* 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.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select or clear (unsigned, the default) this property. If you have set *Physical Type* to `Float`, this is selected. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a float element containing `1234` with a Virtual Decimal value of `3` is `1.234`. This property is not applicable if you have set *Physical Type* to `Float`.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

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.

CWF properties for attribute reference and local attribute integer types:

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, unsignedShort

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `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. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you have set *Physical Type* to `Packed Decimal`, enter a value between 1 and 6. • If you have set *Physical Type* to `Extended Decimal`, enter a value between 1 and 11.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

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.

CWF properties for attribute reference and local attribute string types:

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, 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 from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element 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. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two 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. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. The default is Fixed Length String.
Length	Button and Integer	If you have selected a *Physical Type* of `Fixed Length String` or `Binary`, and have selected the length to be defined by *Length*, 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).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 110.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong

Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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:

XML wire format properties for attribute reference, element reference, local attribute, and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

XML properties for attribute reference, element reference, local attribute, local element Boolean types:

XML wire format properties for attribute reference, element reference, local attribute and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element dateTime types:

XML wire format properties for attribute reference, element reference, local attribute, and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

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 about the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 790. See "DateTime formats" on page 782 for details of dateTime formats.

XML properties for attribute reference, element reference, local attribute, local element decimal types:

XML wire format properties for attribute reference, element reference, local attribute, and 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, unsignedLong

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element float types:

XML wire format properties for attribute reference, element reference, local attribute, and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element integer types:

XML wire format properties for attribute reference, element reference, local attribute, and 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, unsignedShort

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element string types:

XML wire format properties for attribute reference, element reference, local attribute, and 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, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Local attribute TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong

Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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 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	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. The default is dependent on the setting of the message set property Derive default length from logical type. If Derive default length from logical type is selected, the default value is derived from any length or maxLength value constraint (schema facet) on the object's simple type.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

TDS properties for local attribute Boolean types:

The TDS format 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	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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.

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Binary. The data is in bit string format. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.

TDS properties for local attribute dateTime types:

The TDS format 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	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Length Encoded String 1. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a Length Encoded String 1 element is 255 length units. • Length Encoded String 2. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a Length Encoded String 2 element is 65535 length units. • Null Terminated String. The data string ends with the hexadecimal NULL character, `X'00'`. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • Binary. The data is in bit string format. • Time Seconds. This value supports C time_t, and Java Date and Time objects. It is valid only if the DateTime Format property represents numeric-only data. • Time Milliseconds. This value supports C time_t, and Java Date and Time objects. It is valid only if the DateTime Format property represents numeric-only data. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.

Property	Type	Meaning
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
DateTime Format	String	Specify a template for date and time. The default DateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 790. See “DateTime formats” on page 782 for details of date and time formats.

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the Physical type property is Packed Decimal. By default, this check box is cleared, which indicates that the value is not signed.

TDS properties for local attribute decimal types:

The TDS format 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, unsignedLong

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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.

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • TLOG Specific - this option can be selected only if the Message Standard property of the message set is TLOG. This option indicates that the format of the data is specific to the TLOG messaging standard. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.

Property	Type	Meaning
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the Precision property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.
Precision	Button and Integer	This property is applicable only if Physical Type is Text. It is used if the value of the Virtual Decimal Point property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set Precision to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123. • If you set Precision 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 Precision to 2. • If you set Precision 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 Precision to 5. Or select one option from the list: • All Significant Digits - decimal separator only required if fractional digits (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • Explicit Decimal Separator - decimal separator always required: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present.

TDS properties for local attribute float types:

The TDS format 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	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Float. Equates to the data type FLOAT or DOUBLE in C, or the COMP-1 or COMP-2 numeric data type in COBOL. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the Precision property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if Physical Type is Text. It is used if the value of the Virtual Decimal Point property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set Precision to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set Precision 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 Precision to 2. • If you set Precision 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 Precision to 5. Or select one option from the list: • All Significant Digits - decimal separator only required if fractional digits (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • Explicit Decimal Separator - decimal separator always required: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present. • Exponential Notation - Example "1.23456e002": data is written out to the bit stream as a signed value having the format `[sign1]a.bbbe[sign2]ccc` where: – `[sign1]` is the value of Negative Sign if the value is negative – `a` is a single decimal digit – `bbb` is one or more decimal digits – `[sign2]` is the value of Negative Sign if the exponent is negative – `ccc` is exactly three decimal digits (the exponent) `[sign1]` and `[sign2]` are absent if the value and exponent are positive. 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 that the value of Negative Sign is "-", and the value of Sign Orientation is `Leading`. The value -0.00012 is represented as `1.2*e*004` if Negative Sign is "*" and Sign Orientation is Trailing.

TDS properties for local attribute integer types:

The TDS format 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, unsignedShort

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

TDS properties for local attribute string types:

The TDS format properties for local attribute string types.

The TDS Format properties described here apply to:

- Objects: Local Attribute
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.
Interpret Element Value	Enumerated type	Specify whether values stored within this object are interpreted as having significance for the parser and, if so, the type of interpretation that occurs. This interpretation is standard-specific and is therefore hard coded. The possible values for this property are: • None (the default value) • EDIFACT Service String • X12 Service String • Message Key • EDIFACT Syntax Level ID • HL7 Service String • HL7 Field Separator **Note:** The Message Key enumeration has been deprecated.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Length Encoded String 1. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a Length Encoded String 1 element is 255 length units. • Length Encoded String 2. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a Length Encoded String 2 element is 65535 length units. • Null Terminated String. The data string ends with the hexadecimal NULL character, `X'00'`. • TLOG Specific - this option can be selected only if the Message Standard property of the message set is TLOG. This option indicates that the format of the data is specific to the TLOG messaging standard. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.

Property	Type	Meaning
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Local element properties

Different types of properties are available for a local element.

A local element can have the following properties;

- “Local element logical properties” on page 209
- “Local element CWF properties” on page 222
- “Local element XML properties” on page 229
- “Local element TDS properties” on page 246
- “Documentation properties for all message set objects” on page 187

Local element logical properties:

The logical properties of a local element include properties that specify the number of occurrences and value of the local element.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • `-` the hyphen • `_` the underscore • `.` the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.
Type	Enumerated type	The Type property constrains the type of data that can be present in the object. Select from the following options: • `int` • `string` • `Boolean` • `hexBinary` • `dateTime` • `date` • `time` • `decimal` • `float` • `(More...)` • `(New Simple Type)` • `(New Complex Type)` If you select (More...), the Type Selection wizard starts. In this wizard, you can select any of the available types. If you select (New Simple Type), the New Simple Type wizard starts. In this wizard, you can create an Anonymous simple type that is based on an existing type. This type can be created locally or globally. If you select (New Complex Type), the New Complex Type wizard starts. In this wizard, you can create an Anonymous complex type, which can be derived from an existing base type. This type can be created locally or globally. For further information about these types, and examples of their use, see the XML Schema Part 0: Primer. This document is available on the World Wide Web Consortium (W3C) Web site.

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 are displayed in the drop-down list.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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.

Value

Property	Type	Meaning
Default	Button and String	This property provides the default value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.

Property	Type	Meaning
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.
Nillable	Check box	Select this option if you want the element to be able to be defined as null. A null value is distinct from being empty, when the element contains no data.
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. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds 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 remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

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	Limit the set of elements that can belong to its substitution group. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element. • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element. • #all. Prohibit substitution by all methods.
Block	Enumerated type	Limit the set of elements that can be substituted for this element in a message. • Empty • restriction. Prohibit element substitution by elements whose types are restrictions of the type of the head element • extension. Prohibit element substitution by elements whose types are extensions of the type of the head element • substitution. Prohibit element substitution by members of the element's substitution group. • #all. Prohibit substitution by all methods.
Substitution Group	Enumerated type	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 option 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, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong

Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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:

CWF wire format 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	Button and Integer	If you have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1. The maximum value that you can specify is 2147483647. The default value is empty (not set).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 110.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`

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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* 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 CWF wire format 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: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`

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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* 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 CWF wire format 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	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element 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. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two 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. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The dateTime is coded as a Packed Decimal number. It is valid only if the *DateTime Format* property represents numeric-only data. • `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 782 for details). • `Time Seconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *DateTime Format* property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *DateTime Format* property represents numeric-only data. The default value is `fixed length string`.
Length	Button and Integer	If you have selected a *Physical Type* of `Fixed Length String`, `Packed Decimal`, or `Binary`, and have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1 for all three physical types. The maximum value that you can specify is `256` for `Fixed Length String`, `10` for `Packed Decimal`, and `2147483647` for `Binary`. The default value is 0 (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 110.
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.
DateTime Format	String	Specify a template for date and time. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 790. See “DateTime formats” on page 782 for details of date and time formats.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the *Physical type* property is `Packed Decimal`. By default, this check box is cleared, which indicates that the value is not signed.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list. The option that you select determines the value that you must set for the property *Encoding Null Value*: • `NULLPadFill`. This option is valid only if *Physical Type* is `Fixed Length String`. The field is filled with the value specified by the *Padding Character*. The default value. • `NULLLogicalValue`. The *Encoding Null Value* property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This specifies that *Encoding Null Value* contains a value that is directly substituted as if it is a string. Use this option when the value you have set for *Encoding Null Value* to specify a null date is not a dateTime value, or does not conform to the standard dateTime format `yyyy-MM-dd 'T'HH:mm:ss`. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.
Encoding Null Value	String	If you set the *Encoding Null* property to `NULLPadFill`, this property is disabled. If you set the *Encoding Null* property to `NULLLogicalValue`, you must set this property to an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value conforming to `yyyy-MM-dd'T'HH:mm:ss` such as `1970-12-01`. If you set the *Encoding Null* property to `NULLLiteralValue`, you can enter any value that is the same length as the field. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* 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 CWF wire format 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, unsignedLong

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `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. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *Physical Type* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *Physical Type* 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.)
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.

Property	Type	Meaning
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a decimal element containing `1234` with a Virtual Decimal value of `3` is `1.234`, equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This option is valid only if *Physical Type* is `External Decimal`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for `NULLLiteralFill`. The default value is zero. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed list • Enter a character between quotation marks, for example `'c'` or `"c"`, where `c` is any alphanumeric character. • Enter a hexadecimal character 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for element reference and local element float types:

The CWF wire format 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	Select one from the displayed list: • `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. • `Float`. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL and is the default value. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer`, `Packed Decimal`, and `Float` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Float`, select a value from the displayed list. The default value is 8. • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *Physical Type* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *Physical Type* 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.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select or clear (unsigned, the default) this property. If you have set *Physical Type* to `Float`, this is selected. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a float element containing `1234` with a Virtual Decimal value of `3` is `1.234`. This property is not applicable if you have set *Physical Type* to `Float`.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This option is valid only if *Physical Type* is `External Decimal`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for `NULLLiteralFill`. The default value is zero. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed list • Enter a character between quotation marks, for example `'c'` or `"c"`, where `c` is any alphanumeric character. • Enter a hexadecimal character 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for element reference and local element integer types:

The CWF wire format 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, unsignedShort

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `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. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you have set *Physical Type* to `Packed Decimal`, enter a value between 1 and 6. • If you have set *Physical Type* to `Extended Decimal`, enter a value between 1 and 11.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Numeric representation

Property	Type	Meaning
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This option is valid only if *Physical Type* is `External Decimal`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for `NULLLiteralFill`. The default value is zero. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed list • Enter a character between quotation marks, for example `'c'` or `"c"`, where `c` is any alphanumeric character. • Enter a hexadecimal character 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for element reference and local element string types:

The CWF wire format 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, 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 from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element 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. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two 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. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. The default is Fixed Length String.
Length	Button and Integer	If you have selected a *Physical Type* of `Fixed Length String` or `Binary`, and have selected the length to be defined by *Length*, 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).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 110.
Inclusive Length Reference	Check box	This property is applicable only if *Length Reference* is set. If the check box is selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by *Length Reference* is the length of the current object only. If the check box is selected, the *Length Units* property of the sibling integer object must be the same as that of the current object.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This is valid only if *Physical Type* is `Fixed Length String`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.
Encoding Null Value	STRING	The use of this property depends on the *Encoding Null* property. If specified, its length must be equal to the length of the string element, except for `NULLLiteralFill`. The default value is empty (not set). If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Local element XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong
Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: 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:

XML wire format properties for attribute reference, element reference, local attribute, and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

XML properties for attribute reference, element reference, local attribute, local element Boolean types:

XML wire format properties for attribute reference, element reference, local attribute and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element dateTime types:

XML wire format properties for attribute reference, element reference, local attribute, and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

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 about the defaults for the dateTime format according to the logical type see "DateTime defaults by logical type" on page 790. See "DateTime formats" on page 782 for details of dateTime formats.

XML properties for attribute reference, element reference, local attribute, local element decimal types:

XML wire format properties for attribute reference, element reference, local attribute, and 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, unsignedLong

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element float types:

XML wire format properties for attribute reference, element reference, local attribute, and 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 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element integer types:

XML wire format properties for attribute reference, element reference, local attribute, and 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, unsignedShort

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML properties for attribute reference, element reference, local attribute, local element string types:

XML wire format properties for attribute reference, element reference, local attribute, and 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, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Local element TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types - base64Binary - hexBinary	Boolean types - Boolean	Complex types
DateTime types - date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	Decimal types - decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong	Float types - double - float

Integer types	Interval types	String types
- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Note: The physical format properties for simple type *duration* are the same as the physical properties of the string logical types.

TDS properties for local element binary types:

The TDS format properties for local element binary types.

The TDS Format properties described here apply to:

- Objects: Local Element
- Binary schema types: base64Binary, hexBinary

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	For all Messaging Standard values, the Physical Type property is set to Binary and cannot be changed.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. The default is dependent on the setting of the message set property Derive default length from logical type. If Derive default length from logical type is selected, the default value is derived from any length or maxLength value constraint (schema facet) on the object's simple type.
Length Units	Enumerated type	Always set to Bytes.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

TDS properties for local element Boolean types:

The TDS format properties for local element Boolean types.

The TDS Format properties described here apply to:

- Objects: Local Element
- Boolean schema types: Boolean

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Binary. The data is in bit string format. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

TDS properties for local element of complex type:

The TDS Format properties that apply to local elements of complex type.

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

TDS properties for local element dateTime types:

The TDS format properties for local element dateTime types.

The TDS Format properties described here apply to:

- Objects: Local Element
- DateTime schema types: date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Length Encoded String 1. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a Length Encoded String 1 element is 255 length units. • Length Encoded String 2. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a Length Encoded String 2 element is 65535 length units. • Null Terminated String. The data string ends with the hexadecimal NULL character, `X'00'`. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • Binary. The data is in bit string format. • Time Seconds. This value supports C time_t, and Java Date and Time objects. It is valid only if the DateTime Format property represents numeric-only data. • Time Milliseconds. This value supports C time_t, and Java Date and Time objects. It is valid only if the DateTime Format property represents numeric-only data. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if one of the following statements is true: • Physical Type is Packed Decimal. • Physical Type is Text, no Length Reference is specified, and the Data Element Separation of the parent complex type or group is Fixed Length, Tagged Fixed Length, or Fixed Length AL3. The default is dependent on the physical type of the object. If Physical Type is Length Encoded String 1, Length Encoded String 2, or Null Terminated String, this property is not applicable. If Physical Type is Time Seconds, the value of this property is 4, and cannot be changed. If Physical Type is Time Milliseconds, the value of this property is 8, and cannot be changed.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
DateTime Format	String	Specify a template for date and time. The default DateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 790. See “DateTime formats” on page 782 for details of date and time formats.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed. This property is applicable only if the Physical type property is Packed Decimal. By default, this check box is cleared, which indicates that the value is not signed.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

TDS properties for local element decimal types:

The TDS format properties for local element decimal types.

The TDS Format properties described here apply to:

- Objects: Local Element
- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • TLOG Specific - this option can be selected only if the Message Standard property of the message set is TLOG. This option indicates that the format of the data is specific to the TLOG messaging standard. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any minInclusive, maxInclusive, minExclusive, or maxExclusive value constraints (schema facets), on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed.
Sign EBCDIC Custom Overpunched	Check box	Specify whether EBCDIC custom sign format is used. This property is applicable only if the Signed property is selected and the Physical Type property is set to External Decimal.
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the Precision property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if Physical Type is Text. It is used if the value of the Virtual Decimal Point property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set Precision to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set Precision 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 Precision to 2. • If you set Precision 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 Precision to 5. Or select one option from the list: • All Significant Digits - decimal separator only required if fractional digits (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • Explicit Decimal Separator - decimal separator always required: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 770.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

TDS properties for local element float types:

The TDS format properties for local element float types.

The TDS Format properties described here apply to:

- Objects: Local Element
- Float schema types: double, float

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Float. Equates to the data type FLOAT or DOUBLE in C, or the COMP-1 or COMP-2 numeric data type in COBOL. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed.
Sign EBCDIC Custom Overpunched	Check box	Specify whether EBCDIC custom sign format is used. This property is applicable only if the Signed property is selected and the Physical Type property is set to External Decimal.
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the Precision property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if Physical Type is Text. It is used if the value of the Virtual Decimal Point property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set Precision to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set Precision 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 Precision to 2. • If you set Precision 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 Precision to 5. Or select one option from the list: • All Significant Digits - decimal separator only required if fractional digits (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • Explicit Decimal Separator - decimal separator always required: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present. • Exponential Notation - Example "1.23456e002": data is written out to the bit stream as a signed value having the format `[sign1]a.bbbe[sign2]ccc` where: – `[sign1]` is the value of Negative Sign if the value is negative – `a` is a single decimal digit – `bbb` is one or more decimal digits – `[sign2]` is the value of Negative Sign if the exponent is negative – `ccc` is exactly three decimal digits (the exponent) `[sign1]` and `[sign2]` are absent if the value and exponent are positive. 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 that the value of Negative Sign is "-", and the value of Sign Orientation is `Leading`. The value -0.00012 is represented as `1.2*e*004` if Negative Sign is "*" and Sign Orientation is Trailing.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see "TDS Null handling options" on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

TDS properties for local element integer types:

The TDS format properties for local element integer types.

The TDS Format properties described here apply to:

- Objects: Local Element
- Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, unsignedShort

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see "Regular expression syntax" on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any minInclusive, maxInclusive, minExclusive, or maxExclusive value constraints (schema facets), on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Property	Type	Meaning
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Numeric representation

Property	Type	Meaning
Signed	Check box	Specify whether the value is signed.
Sign EBCDIC Custom Overpunched	Check box	Specify whether EBCDIC custom sign format is used. This property is applicable only if the Signed property is selected and the Physical Type property is set to External Decimal.
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see "TDS Null handling options" on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

TDS properties for local element interval types:

The TDS format properties for local element string types.

The TDS Format properties described here apply to:

- Objects: Local Element
- Interval schema types: duration

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Length Encoded String 1. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a Length Encoded String 1 element is 255 length units. • Length Encoded String 2. The data's first 2 bytes contains the length (in length units) of the data string that follows the two length bytes. The maximum length of a Length Encoded String 2 element is 65535 length units. • Null Terminated String. The data string ends with the hexadecimal NULL character, `X'00'`. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see "TDS Null handling options" on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

TDS properties for local element string types:

The TDS format properties for local element string types.

The TDS Format properties described here apply to:

- Objects: Local Element
- String schema types: anyURI, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Field Identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.
Interpret Element Value	Enumerated type	Specify whether values stored within this object are interpreted as having significance for the parser and, if so, the type of interpretation that occurs. This interpretation is standard-specific and is therefore hard coded. The possible values for this property are: • None (the default value) • EDIFACT Service String • X12 Service String • Message Key • EDIFACT Syntax Level ID • HL7 Service String • HL7 Field Separator **Note:** The Message Key enumeration has been deprecated.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Length Encoded String 1. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a Length Encoded String 1 element is 255 length units. • Length Encoded String 2. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a Length Encoded String 2 element is 65535 length units. • Null Terminated String. The data string ends with the hexadecimal NULL character, `X'00'`. • TLOG Specific - this option can be selected only if the Message Standard property of the message set is TLOG. This option indicates that the format of the data is specific to the TLOG messaging standard. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Length Units	Enumerated type	Select the unit of length for the object. Select one of the following options (some physical types do not offer both options): • Bytes. The length is given in bytes. • Characters. The length is given in characters. The number of bytes that are processed in the bit stream depends on the code page of the message. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the contents of the bit stream. The parser reads one character at a time and determines whether the character comprises one or more bytes. The default is dependent on the physical type of the object.

Property	Type	Meaning
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.
Inclusive Length Reference	Check box	This property is applicable only if Length Reference is set. If the check box is selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object plus the length of the sibling integer object. If the check box is not selected, the value of the sibling integer object that is identified by Length Reference is the length of the current object only. If the check box is selected, the Length Units property of the sibling integer object must be the same as that of the current object.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

Occurrences

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Local group properties

Different types of properties are available for a local group.

A local group can have the following properties;

- "Local group logical properties" on page 213
- "Local group CWF properties" on page 223
- "Local group XML properties" on page 230
- "Local group TDS properties" on page 247
- "Documentation properties for all message set objects" on page 187

Local group logical properties:

The logical properties of a local group include properties that specify the number of occurrences of the local group.

Valid children in a local group that depend on both **Composition** and **Content Validation** are shown in "MRM content validation" on page 194.

Property	Type	Meaning
Composition	Enumerated type	Define the order, and the number of occurrences, of the elements and groups in your messages. Composition does not affect the attributes in a complex type. Select from the following options: • Empty • sequence. If you select this option, you can define members that are elements or groups. These members, if present, must appear in the specified order in the message. They can repeat, and the same element or group can appear more than once. • choice. If you select this option, you can define members that are elements or groups. Exactly one of the defined members must be present in the message, and can repeat. 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. • all. If you select this option, you can define members that are elements; groups are not allowed. The elements in an all group can appear in any order. Each element can appear once, or not at all. An all group can be used only at the top level of a complex type; it cannot be a member of another group within a type. • unorderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements can appear in any order in the message. • orderedSet. This option is supported only by the MRM domain.If you select this option, you can define members that are elements. The elements can repeat, but the same element cannot appear twice in the list of members. The elements must appear in the specified order in the message. • message. This option is supported only by the MRM domain. If you select this option, you can define only messages as members. Each member can repeat, but the same message cannot appear twice in the list of members. Like choice, only one of the defined members can be present in a message. Unlike choice, when writing a message, if the complex type or group has more than one member, the bit stream is not padded to the length of the longest member. 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 26.

Property	Type	Meaning
Content Validation	Enumerated type	Content Validation is used only by the MRM domain. If validation is enabled in your message flow, Content Validation specifies the strictness of the MRM validation for members of a complex type or group. See “MRM content validation” on page 194 for further details. Select from the following options: • Closed. The complex type can only contain the child elements that you have added to it. • Open Defined. The complex type can contain any valid element defined within the message set. • Open. The complex type can contain any valid element, not just those that you have added to this complex type. See “Combinations of Composition and Content Validation” on page 300 for further details of these options.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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.

Local group CWF properties:

The CWF properties of a local group are described in the following tables.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`

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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Local group XML properties:

There are no properties to show.

Local group TDS properties:

TDS properties of a local group.

Field Identification

Property	Type	Meaning
Data Element Separation	Enumerated type	Select one of the following values to specify the method that is used to separate the data elements within the type. • Tagged Delimited. This value indicates that all elements within the complex type are identified by a tag, and, if a value is specified in the optional Delimiter property, are separated by that value. You must set the Tag property for all child elements of simple type, and you can set the Delimiter property to a non-empty value. See "Global element TDS properties" on page 240. You must also set the Tag Data Separator or the Length of Tag property. • 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 Tag property for each of the child elements of this complex type, and each child element must have a Length or Length Reference property assigned to it. You must also set the Tag Data Separator or the Length of Tag property. • 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 the Length of Tag property, or variable length delimited by the Tag Data Separator property. You must also set the Length Of Encoded Length property so that the parser knows the size of the length field, and set the Extra Chars in Encoded Length property. This property tells the parser what to subtract from the value in the Length Of Encoded Length property to get the actual length of the data that follows the length field. This method provides a more flexible way of handling ACORD AL3 standard messages than using the Fixed Length AL3 value, by allowing different parts of the messages to be at different versions of the ACORD AL3 standard. • All Elements Delimited. This value indicates that all elements within the complex type are separated by a delimiter. You must set a value in the Delimiter property. • Variable Length Elements Delimited. This value indicates that some of the elements within the complex type might be of variable length. Variable length elements must be delimited by the value specified in the Delimiter property. • Use Data Pattern. This value indicates that the parser determines the elements by matching the data with the regular expression that is set in the Data Pattern property of the element or type member. See "Message definition file properties" on page 185. • 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 Length property to the offset. See "Global element TDS properties" on page 240. If you set the Data Element Separation property of a complex type to Fixed Length, you must also set the Data Element Separation property of all complex children of this type to Fixed Length. Each child element must have a Length or Length Reference property assigned to it. • 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 regarding missing optional elements, encoded lengths, and versioning, must be applied. If you set the Data Element Separation property of a complex type to Fixed Length AL3, you must also set the Data Element Separation property of all complex children of this type to Fixed Length AL3. • Undefined. This value is set automatically if you set the Type Composition property of a complex type to Message, and you cannot change it to any other value. Do not set the Type Composition property to Empty, Choice, Unordered Set, Ordered Set, Sequence, or Simple Unordered Set. If you do, you cannot check in the type.

Property	Type	Meaning
Group Indicator	String	Specify the value of a special character, or string, that precedes the data that belongs to a group, or a complex type, within the bit stream.
Group Terminator	String	Specify the value of a special character, or string, that terminates the data that belongs 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 that is used between data elements. This property applies only to the delimited Data Element Separation methods (Tagged Delimited, All Elements Delimited, and Variable Length Elements Delimited).
Suppress Absent Element Delimiters	Enumerated type	Use this property to select whether you want delimiters to be suppressed for elements that are missing within a message. Select from: • 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 three elements and only two are present, the last delimiter can be omitted from the message. • Never. Use this option to ensure that even if optional elements are not present, all delimiters are written out. Use this option when the same delimiter is used to delimit parent and child objects. For example, if an optional child element is missing, message processing applications cannot tell where the child elements in a message end and the next parent element starts, if the delimiters are all the same.
Observe Element Length	Check box	This property is applicable when Data Element Separation is All Elements Delimited or Tagged Delimited. Select this check box if the Length property of child simple elements is significant when parsing and writing. • During parsing, an exception is thrown if the length of the extracted data exceeds the specified length. Otherwise, the data is trimmed according to the Justification and Padding Character properties of the child element. • During writing, an exception is thrown if the data to write exceeds the specified length. Otherwise, the data is padded according to the Justification and Padding Character properties of the child element. Clear this check box to ignore the Length property when parsing and writing. The default value depends on the setting of the Messaging Standard property (at the message set level) and the Data Element Separation property. • If Data Element Separation is All Elements Delimited and the Messaging Standard is TLOG, the check box is selected. • If Data Element Separation is All Elements Delimited and the Messaging Standard is other than TLOG, the check box is cleared. • If Data Element Separation is Tagged Delimited, the check box is cleared. For all other data element separation methods, the check box is disabled and does not influence the behavior of the TDS parser.
Tag Data Separator	Button and String	Specify the value of a special character or string that separates the Tag from the data. The Tag Data Separator and Length of Tag properties are mutually exclusive. If you set the property Tag Data Separator, it overrides Length of Tag. This property applies only to the tagged Data Element Separation methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).

Property	Type	Meaning
Length of Tag	Button and Integer	Specify the length of a tag value. When the message is parsed, this property allows tags to be extracted from the bit stream if the Tag Data Separator property is not set. The Tag Data Separator and Length of Tag properties are mutually exclusive. If you set the property Tag Data Separator, it overrides this value. This property applies only to the tagged Data Element Separation methods (Tagged Delimited, Tagged Fixed Length, and Tagged Encoded Length).
Length of Encoded Length	Integer	Specifies the number of characters (not bytes) after a tag that are used for the length field. Enter a value from 0 to 2147483647. You must set this property if you have set the Data Element Separation property to Tagged Encoded Length; it is not valid otherwise. The actual number of data characters that are parsed depends on the value of the Extra Chars in Encoded Length property.
Extra Chars in Encoded Length	Integer	(Only valid if the Data Element Separation 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.) 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. You must set this property if you have set the Data Element Separation property to Tagged Encoded Length, and the actual number of data characters is less than the value found in the length field.

Field Identification

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Occurrences

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

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 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 no objects are listed, no integer objects exist before this one in the message structure. If a Repeat Reference is specified, it overrides any setting for the Max Occurs logical property when parsing and writing the message, but not for validating the message.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Message properties

Different types of properties are available for a message.

A message can have the following properties;

- "Message logical properties" on page 215
- "Message CWF properties" on page 224
- "Message XML properties" on page 230
- "Message TDS properties" on page 251
- "Documentation properties for all message set objects" on page 187

Message logical properties:

This section describes the logical properties of a message.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.
Message Alias	String	Specify an alternative unique value that identifies the message. This property is only required if you are using the MRM domain and the Message Identity technique to identify embedded messages, and the bit stream does not contain the actual message name.

Message CWF properties:

There are no properties to show.

Message XML properties:

The following tables describe the XML properties of a message.

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. If namespaces have not been enabled, this property displays <no target namespace>. This property overrides the same property at the message set level.
Schema location	String	Enter the location of the schema for the associated namespace name to be used to validate objects in the namespace.

XML declarations

Property	Type	Meaning
Output Namespace Declaration	Enumerated type	The Output Namespace Declaration property controls where the namespace declarations are placed in the output XML document. Select from: • At start of document. Declarations for all of the entries in the Namespace schema locations table above are produced as attributes of the message in the output XML document. The disadvantage of this option is that in some cases unnecessary declarations might be produced. • As required. Declarations are produced only when required by an element or attribute that is in that namespace. The disadvantage of this option is that the same namespace declaration might need to be produced more than once in the output XML document. The default option is At start of document. This property is active only if namespaces are enabled for this message set.

XML document type settings

Property	Type	Meaning
DOCTYPE System ID	String	Specify the System ID for DOCTYPE external DTD subset. It overrides the equivalent message set property setting for this particular message. If the message set property Suppress DOCTYPE is set to Yes, this parameter is ignored and cannot be changed (the field is disabled). The default value is the value that you specified for the DOCTYPE System ID property for the message set.
DOCTYPE Public ID	String	Specify the Public ID for DOCTYPE external DTD subset. It overrides the equivalent message set property setting for this particular message. If the message set property Suppress DOCTYPE is set to Yes, this parameter is ignored and cannot be changed (the field is disabled) . The default value is the value that you specified for the DOCTYPE Public ID property for the message set.
DOCTYPE Text	String	Enter optional additional text to include within the DOCTYPE. It overrides the message set property for this particular message. If the message set property Suppress DOCTYPE is set to Yes, this parameter is ignored and cannot be changed (the field is disabled). For more information, see “MRM XML: In-line DTDs and the DOCTYPE text property” on page 184. The default value is the value that you specified for the DOCTYPE Text property for the message set.

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 Root Tag Name property for the message set. **Note:** This property is deprecated. Do not change its value from its default setting.

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 (for output) in the resulting XML document. Select one of the following values from the drop-down list: • XMLElement. 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 XML Name property to the same value, both objects must refer to the same element. This is the default value for element objects. • XMLElementAttrID. 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 ID Attribute Name and a value as specified in ID Attribute Value. If you select this value for one object, and set this same value or the value XMLElementAttrIDVal for a second object, and set XML Name, ID Attribute Name, ID Attribute Value to the same values: – You must also set Value Attribute Name to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.

Message TDS properties:

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. **Note:** From Version 6.0 onwards, the use of Message Key has been deprecated for identifying an embedded message. You now have the option of identifying an embedded message by Message Identity, using the Message Alias logical property.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Simple type properties

Different types of properties are available for a simple type.

A simple type can have the following properties;

- "Simple type logical properties" on page 215
- "Simple type CWF properties" on page 224
- "Simple type XML properties" on page 233
- "Simple type TDS properties" on page 251
- "Documentation properties for all message set objects" on page 187

A simple type can also have "Simple type logical value constraints" on page 216.

Simple type logical properties:

The logical properties of a simple type.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.
Base Type	Enumerated type	This property only applies to a simple type restriction. 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.
Item Type	Enumerated type	This property only applies to a simple type list. You can use this property to select the type that is used as the item type of the list.
Variety	Enumerated type	This property displays the variety of the simple type you have selected, either atomic, list, or union.

A simple type can also have "Simple type logical value constraints" on page 216.

Simple type logical value constraints:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- base64Binary - hexBinary	- Boolean	- date - dateTime - gDay - gMonth - gMonthDay - gYear - gYearMonth - time	- decimal - integer - negativeInteger - nonNegativeInteger - nonPositiveInteger - positiveInteger - unsignedLong

Float types	Integer types	Interval types	String types
- double - float	- byte - int - long - short - unsignedByte - unsignedInt - unsignedShort	- duration	- anyURI - ENTITIES - ENTITY - ID - IDREF - IDREFS - language - Name - NCName - NMTOKEN - NMTOKENS - normalizedString - NOTATION - QName - string - token

Logical properties for value constraints for simple type binary types:

The logical properties 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	Specify the exact length of the simple type in bytes or characters. The value must be greater than 0, and less than 2147483648.
Min	Integer	Specify the minimum length of the simple type in bytes or characters. The value must be greater than 0, and less than 2147483648.
Max	Integer	Specify the maximum length of the simple type in bytes or characters. The value must be greater than 0, and less than 2147483648.

Property	Type	Meaning
White Space	Enumerated type	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Enumerations

Property	Type	Meaning
Enumerations	String	Set this property to constrain the values to the list that is specified in this property. For example, you might create a simple type called *RainbowColors*, and add Red, `Orange`, `Yellow`, `Green`, `Blue`, `Indigo`, and `Violet` to the enumerations list. You must 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, and y might be variations of a single color. Select **Add** to add a default enumeration. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Patterns

Property	Type	Meaning
Patterns	String	Patterns are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see "Using regular expressions to parse data elements" on page 776. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Logical properties for value constraints for simple type Boolean types:

The logical properties 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	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Patterns

Property	Type	Meaning
Patterns	String	Patterns are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see "Using regular expressions to parse data elements" on page 776. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Logical properties for value constraints for simple type dateTime types:

The logical properties 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	Specify the minimum value for which the data in the message must be greater than, or equal to. When this value is set, it cannot be equal to, or greater than, the Max Inclusive Constraint property. You cannot specify both Min Inclusive Constraint and Min Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than, or equal to. When this value is set, it cannot be equal to, or less than, the Min Inclusive Constraint property. You cannot specify both Max Inclusive Constraint and Max Exclusive Constraint properties together for the same simple type.

Exclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than. When this value is set it cannot be equal to, or greater than, the Max Inclusive Constraint property. You cannot specify both Min Inclusive Constraint and Min Exclusive Constraint properties together for the same simple type.

Property	Type	Meaning
Max	Integer	Specify the maximum value for which the data in the message must be less than. When this value is set, it cannot be equal to, or less than, the Min Inclusive Constraint property. You cannot specify both Max Inclusive Constraint and Max Exclusive Constraint properties together for the same simple type.

Property	Type	Meaning
White Space	Enumerated type	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Enumerations

Property	Type	Meaning
Enumerations	String	Set this property to constrain the values to the list that is specified in this property. For example, you might create a simple type called *RainbowColors*, and add Red, `Orange`, `Yellow`, `Green`, `Blue`, `Indigo`, and `Violet` to the enumerations list. You must 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, and y might be variations of a single color. Select **Add** to add a default enumeration. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Patterns

Property	Type	Meaning
Patterns	String	Patterns are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see “Using regular expressions to parse data elements” on page 776. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Logical properties for value constraints for simple type decimal types:

The logical properties 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, unsignedLong

Inclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than, or equal to. When this value is set, it cannot be equal to, or greater than, the Max Inclusive Constraint property. You cannot specify both Min Inclusive Constraint and Min Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than, or equal to. When this value is set, it cannot be equal to, or less than, the Min Inclusive Constraint property. You cannot specify both Max Inclusive Constraint and Max Exclusive Constraint properties together for the same simple type.

Exclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than. When this value is set it cannot be equal to, or greater than, the Max Inclusive Constraint property. You cannot specify both Min Inclusive Constraint and Min Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than. When this value is set, it cannot be equal to, or less than, the Min Inclusive Constraint property. You cannot specify both Max Inclusive Constraint and Max Exclusive Constraint properties together for the same simple type.

Property	Type	Meaning
Fraction Digits	Integer	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. The value must be greater than, or equal to, 0, and less than 2147483648. The value set for Fraction Digits cannot be greater than the value specified for Total Digits.

Property	Type	Meaning
Total Digits	Integer	Set this property to set the maximum number of digits in a numerical value to the number of digits specified in this property. The value must be greater than, or equal to, 0, and less than 2147483648. The value set for Total Digits cannot be less than the value specified for Fraction Digits.

Property	Type	Meaning
White Space	Enumerated type	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Enumerations

Property	Type	Meaning
Enumerations	String	Set this property to constrain the values to the list that is specified in this property. For example, you might create a simple type called *RainbowColors*, and add Red, `Orange`, `Yellow`, `Green`, `Blue`, `Indigo`, and `Violet` to the enumerations list. You must 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, and y might be variations of a single color. Select **Add** to add a default enumeration. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Patterns

Property	Type	Meaning
Patterns	String	Patterns are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see “Using regular expressions to parse data elements” on page 776. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Logical properties for value constraints for simple type float types:

The logical properties 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	Specify the minimum value for which the data in the message must be greater than, or equal to. When this value is set, it cannot be equal to, or greater than, the Max Inclusive Constraint property. You cannot specify both Min Inclusive Constraint and Min Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than, or equal to. When this value is set, it cannot be equal to, or less than, the Min Inclusive Constraint property. You cannot specify both Max Inclusive Constraint and Max Exclusive Constraint properties together for the same simple type.

Exclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than. When this value is set it cannot be equal to, or greater than, the Max Inclusive Constraint property. You cannot specify both Min Inclusive Constraint and Min Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than. When this value is set, it cannot be equal to, or less than, the Min Inclusive Constraint property. You cannot specify both Max Inclusive Constraint and Max Exclusive Constraint properties together for the same simple type.

Property	Type	Meaning
White Space	Enumerated type	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Enumerations

Property	Type	Meaning
Enumerations	String	Set this property to constrain the values to the list that is specified in this property. For example, you might create a simple type called *RainbowColors*, and add Red, `Orange`, `Yellow`, `Green`, `Blue`, `Indigo`, and `Violet` to the enumerations list. You must 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, and y might be variations of a single color. Select **Add** to add a default enumeration. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Patterns

Property	Type	Meaning
Patterns	String	Patterns are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see “Using regular expressions to parse data elements” on page 776. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Logical properties for value constraints for simple type integer types:

The logical properties 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, unsignedShort

Inclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than, or equal to. When this value is set, it cannot be equal to, or greater than, the Max Inclusive Constraint property. You cannot specify both Min Inclusive Constraint and Min Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than, or equal to. When this value is set, it cannot be equal to, or less than, the Min Inclusive Constraint property. You cannot specify both Max Inclusive Constraint and Max Exclusive Constraint properties together for the same simple type.

Exclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than. When this value is set it cannot be equal to, or greater than, the Max Inclusive Constraint property. You cannot specify both Min Inclusive Constraint and Min Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than. When this value is set, it cannot be equal to, or less than, the Min Inclusive Constraint property. You cannot specify both Max Inclusive Constraint and Max Exclusive Constraint properties together for the same simple type.

Property	Type	Meaning
Fraction Digits	Integer	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. The value must be greater than, or equal to, 0, and less than 2147483648. The value set for Fraction Digits cannot be greater than the value specified for Total Digits.

Property	Type	Meaning
Total Digits	Integer	Set this property to set the maximum number of digits in a numerical value to the number of digits specified in this property. The value must be greater than, or equal to, 0, and less than 2147483648. The value set for Total Digits cannot be less than the value specified for Fraction Digits.

Property	Type	Meaning
White Space	Enumerated type	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Enumerations

Property	Type	Meaning
Enumerations	String	Set this property to constrain the values to the list that is specified in this property. For example, you might create a simple type called *RainbowColors*, and add Red, `Orange`, `Yellow`, `Green`, `Blue`, `Indigo`, and `Violet` to the enumerations list. You must 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, and y might be variations of a single color. Select **Add** to add a default enumeration. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Patterns

Property	Type	Meaning
Patterns	String	Patterns are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see "Using regular expressions to parse data elements" on page 776. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Logical properties for value constraints for simple type interval types:

The logical properties 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	Specify the minimum value for which the data in the message must be greater than, or equal to. When this value is set, it cannot be equal to, or greater than, the Max Inclusive Constraint property. You cannot specify both Min Inclusive Constraint and Min Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than, or equal to. When this value is set, it cannot be equal to, or less than, the Min Inclusive Constraint property. You cannot specify both Max Inclusive Constraint and Max Exclusive Constraint properties together for the same simple type.

Exclusive Constraints

Property	Type	Meaning
Min	Integer	Specify the minimum value for which the data in the message must be greater than. When this value is set it cannot be equal to, or greater than, the Max Inclusive Constraint property. You cannot specify both Min Inclusive Constraint and Min Exclusive Constraint properties together for the same simple type.
Max	Integer	Specify the maximum value for which the data in the message must be less than. When this value is set, it cannot be equal to, or less than, the Min Inclusive Constraint property. You cannot specify both Max Inclusive Constraint and Max Exclusive Constraint properties together for the same simple type.

Property	Type	Meaning
White Space	Enumerated type	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Enumerations

Property	Type	Meaning
Enumerations	String	Set this property to constrain the values to the list that is specified in this property. For example, you might create a simple type called *RainbowColors*, and add Red, `Orange`, `Yellow`, `Green`, `Blue`, `Indigo`, and `Violet` to the enumerations list. You must 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, and y might be variations of a single color. Select **Add** to add a default enumeration. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Patterns

Property	Type	Meaning
Patterns	String	Patterns are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see "Using regular expressions to parse data elements" on page 776. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Logical properties for value constraints for simple type string types:

The logical properties 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, unsignedShort

Length constraints

Property	Type	Meaning
Length	Integer	Specify the exact length of the simple type in bytes or characters. The value must be greater than 0, and less than 2147483648.
Min	Integer	Specify the minimum length of the simple type in bytes or characters. The value must be greater than 0, and less than 2147483648.
Max	Integer	Specify the maximum length of the simple type in bytes or characters. The value must be greater than 0, and less than 2147483648.

Property	Type	Meaning
White Space	Enumerated type	Set this property to control the processing of white space characters received for this type. Select one of the following values: • preserve. If you set the property to preserve, all white space characters including carriage return, line feed, and tab are preserved. • replace. If you set the property to replace, all carriage return, line feed, and tab characters are replaced with a space character. • collapse. If you set the property to collapse, all carriage return, line feed, and tab characters are replaced with a space character. All adjacent white space characters are then collapsed to a single space character, and leading or trailing spaces are stripped from the data.

Enumerations

Property	Type	Meaning
Enumerations	String	Set this property to constrain the values to the list that is specified in this property. For example, you might create a simple type called *RainbowColors*, and add Red, `Orange`, `Yellow`, `Green`, `Blue`, `Indigo`, and `Violet` to the enumerations list. You must 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, and y might be variations of a single color. Select **Add** to add a default enumeration. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

Patterns

Property	Type	Meaning
Patterns	String	Patterns are a regular expression, or a series of regular expressions, that are used to constrain the data within the simple type. For further information about patterns and their syntax see "Using regular expressions to parse data elements" on page 776. Select **Add** to add a default pattern. Overtype the default with the data you require. To change an entry, select the entry, and click on the entry a second time (as distinct from double-click). You can now update the selected entry.

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:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Unique properties

Unique logical, CWF, XML, TDS, and documentation properties.

A unique can have the following properties;

- "Unique logical properties" on page 216
- "Unique CWF properties" on page 224
- "Unique XML properties" on page 233
- "Unique TDS properties" on page 251
- "Documentation properties for all message set objects" on page 187

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:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Wildcard attribute properties

Different types of properties are available for a wildcard attribute.

A wildcard attribute can have the following properties;

- "Wildcard attribute logical properties" on page 217

- "Wildcard attribute CWF properties" on page 224
- "Wildcard attribute XML properties" on page 233
- "Wildcard attribute TDS properties" on page 252
- "Documentation properties for all message set objects" on page 187

Wildcard attribute logical properties:

The logical properties of a wildcard attribute.

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, Process Content defines how the attribute is validated. Select one of the following options: • strict. The parser can match only against attributes declared in the specified namespace. • lax. The parser attempts to match against attributes declared in all accessible namespaces. If the specified namespace cannot be found, an error is not generated. • skip. If you select skip, the parser does not perform validation on the attribute.

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:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Wildcard element properties

Different types of properties are available for a wildcard element.

A wildcard element can have the following properties;

- "Wildcard element logical properties" on page 217
- "Wildcard element CWF properties" on page 224
- "Wildcard element XML properties" on page 233
- "Wildcard element TDS properties" on page 252
- "Documentation properties for all message set objects" on page 187

Wildcard element logical properties:

The logical properties of a wildcard element include properties that specify the number of occurrences of the wildcard element.

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 element that corresponds to a wildcard in the message model, Process Content defines how the element is validated. Select one of the following options: • strict. The parser can match only against elements declared in the specified namespace. • lax. The parser attempts to match against elements declared in any accessible namespace. If the specified namespace cannot be found, an error is not generated. • skip. If you select skip the parser does not perform validation on the element.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs. This property is ignored by brokers that are earlier than WebSphere Message Broker Version 6.0.

Property	Type	Meaning
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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 earlier than 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:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Deprecated message model object properties

Some objects in the message model are deprecated, but you can reference the information for their properties.

You can access the reference information for the properties of deprecated message model objects in two ways. The following topics allow you to access the property information by property kind:

- "Logical properties for deprecated message model objects" on page 616
- "Physical properties for deprecated message model objects" on page 619
- "Documentation properties for all message set objects" on page 187

Alternatively, you can access the property information by object, starting from the following topic:

- "Deprecated message model object properties by object" on page 624

Logical properties for deprecated message model objects

Logical property information for compound elements and embedded simple types.

Logical property information is available for the following deprecated objects:

- "Compound element logical properties"
- "Embedded simple type logical properties" on page 619

Compound element logical properties

The logical properties of a compound element include properties that specify the number of occurrences of the compound element.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.
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 are displayed in the drop-down list.
Nillable	Check box	Select this option if you want the element to be able to be defined as null. A null value is distinct from being empty, when the element contains no data.
Abstract	Check box	Select this option 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 an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.

Property	Type	Meaning
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. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds 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 remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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.

Compound element complex type logical properties:

The **Name** property is set to **ANONYMOUS** and cannot be changed.

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 **ANONYMOUS**, 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 value constraints for a compound element.

The properties for compound element value constraints are identical to simple type value constraints. See "Simple type logical value constraints" on page 216 for details.

Embedded simple type logical properties

The logical properties of an embedded simple type include properties that specify the number of occurrences of the embedded simple type.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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

CWF, XML, and TDS format physical properties for deprecated 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 621
- "TDS format physical properties for deprecated objects" on page 622

Custom Wire Format properties for deprecated message model objects

CWF properties for compound elements and embedded simple types.

Custom wire format physical property information is available for the following deprecated objects:

- "Compound element CWF properties"
- "Embedded simple type CWF properties"

Compound element CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

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, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- ComIbmMrm _AnonBinary	- ComIbmMrm _AnonBoolean	- ComIbmMrm _AnonDate - ComIbmMrm _AnonDateTime - ComIbmMrm _AnonGDay - ComIbmMrm _AnonGMonth - ComIbmMrm _AnonGMonthDay - ComIbmMrm _AnonGYear - ComIbmMrm _AnonGYearMonth - ComIbmMrm _AnonTime	- ComIbmMrm _AnonDecimal
Float types	Integer types	String types	
- ComIbmMrm _AnonFloat	- ComIbmMrm _AnonInt	- ComIbmMrm _AnonString	

XML wire format physical properties for deprecated message model objects

XML wire format physical properties for compound elements and embedded simple types.

XML wire format physical property information is available for the following deprecated objects:

- "Compound element XML properties"
- "Embedded simple type XML properties" on page 622

Compound element XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- ComIbmMrm _BaseValueBinary	- ComIbmMrm _BaseValueBoolean	- ComIbmMrm _BaseValueDateTime	- ComIbmMrm _BaseValueDecimal
Float types	Integer types	String types	
- ComIbmMrm _BaseValueFloat	- ComIbmMrm _BaseValueInt	- ComIbmMrm _BaseValueString	

Compound element complex type XML properties:

There are no properties to show.

Embedded simple type XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- ComIbmMrm _AnonBinary	- ComIbmMrm _AnonBoolean	- ComIbmMrm _AnonDate - ComIbmMrm _AnonDateTime - ComIbmMrm _AnonGDay - ComIbmMrm _AnonGMonth - ComIbmMrm _AnonGMonthDay - ComIbmMrm _AnonGYear - ComIbmMrm _AnonGYearMonth - ComIbmMrm _AnonTime	- ComIbmMrm _AnonDecimal
Float types	Integer types	String types	
- ComIbmMrm _AnonFloat	- ComIbmMrm _AnonInt	- ComIbmMrm _AnonString	

TDS format physical properties for deprecated objects

TDS format physical properties for compound elements and embedded simple types.

TDS format physical property information is available for the following deprecated objects:

- "Compound element TDS properties"
- "Embedded simple type TDS properties" on page 623

Compound element TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- ComIbmMrm _BaseValueBinary	- ComIbmMrm _BaseValueBoolean	- ComIbmMrm _BaseValueDateTime	- 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 234 for details.

Embedded simple type TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

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

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Deprecated message model object properties by object

Properties for compound elements and embedded simple types.

The following deprecated objects have properties that can be viewed or set:

- "Compound element properties"
- "Embedded simple type properties" on page 692

Compound element properties

Logical, CWF, XML, TDS, and document properties for a compound element.

- "Compound element logical properties" on page 616
- "Compound element CWF properties" on page 620
- "Compound element XML properties" on page 621
- "Compound element TDS properties" on page 622
- "Documentation properties for all message set objects" on page 187

Compound element logical properties:

The logical properties of a compound element include properties that specify the number of occurrences of the compound element.

Property	Type	Meaning
Name	String	Specify a name for the object when you create it. Names can consist of alphanumeric characters, including the letters *A* through *Z*, *a* through *z*, and the digits *0* through *9*. They might also include the following punctuation characters; • - the hyphen • _ the underscore • . the period Names can start only with a letter or the underscore character, and not with a number, hyphen, or period. Names that begin with `xml`, or any variant of these characters (for example `XmL`), are reserved by the XML standards specification. 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.
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 are displayed in the drop-down list.
Nillable	Check box	Select this option if you want the element to be able to be defined as null. A null value is distinct from being empty, when the element contains no data.
Abstract	Check box	Select this option 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 an element or attribute. **XMLNSC domain** When parsing with validation enabled, default values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if they have default values. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the default value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for default values **Other domains** No support for default values.
Fixed	Button and String	This property provides the fixed value for an element or attribute. **XMLNSC domain** When parsing with validation enabled, if an attribute or element is present, the value is validated against the fixed value. If the values are not equal, a validation error is signalled. Also, when parsing with validation enabled, fixed values are applied to missing attributes and empty elements as required by the XML Schema 1.0 specification. When writing, elements or attributes that are missing from the message tree are not automatically added to the output XML bit stream, even if a fixed value has been specified. If missing elements or attributes are required, the message tree can be serialized and then re-parsed with validation enabled. **MRM (CWF and TDS physical formats)** When writing a fixed-length portion of a message (CWF or fixed-length TDS), if an attribute or element is missing from the message tree, the fixed value is inserted into the bit stream so that the message structure is preserved. **MRM (XML physical format)** No support for fixed values **Other domains** No support for fixed values.

Property	Type	Meaning
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. Select one of the following options: • None This value is the default value, and indicates that the element or attribute does not have a key value associated with it. • MessageSetIdentity. Specifies that the value of the element or attribute corresponds 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 remains in force unless a new element or attribute MessageSetIdentity field is encountered which resets the MessageSetIdentity value. • MessageIdentity. Specifies that the value of the element or attribute corresponds to the name or alias (in that priority order) that is associated with a message, and acts as an identifier for subsequent embedded messages which are the immediate children of the enclosing message. This identity applies until a new element or attribute MessageIdentity field is encountered at the same level in the tree. The embedded message can be defined in either the current message set, or in a message set identified by using a MessageSetIdentity. **Note:** This property is applicable only when the type of the object is derived from xsd:string.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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.

Compound element value constraint properties:

The value constraints for a compound element.

The properties for compound element value constraints are identical to simple type value constraints. See “Simple type logical value constraints” on page 216 for details.

Compound element CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

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:

Physical representation, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

Physical representation

Property	Type	Meaning
Length	Button and Integer	If you have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1. The maximum value that you can specify is `2147483647`. The default value is empty (not set).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 110.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for compound element Boolean types:

Byte alignment and occurrences.

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: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for compound element dateTime types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element 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. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two 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. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The dateTime is coded as a Packed Decimal number. It is valid only if the *DateTime Format* property represents numeric-only data. • `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 782 for details). • `Time Seconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *DateTime Format* property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *DateTime Format* property represents numeric-only data. The default value is `fixed length string`.
DateTime Format	String	Specify a template for date and time. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see "DateTime defaults by logical type" on page 790. See "DateTime formats" on page 782 for details of date and time formats.
Length	Button and Integer	If you have selected a *Physical Type* of `Fixed Length String`, `Packed Decimal`, or `Binary`, and have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1 for all three physical types. The maximum value that you can specify is `256` for `Fixed Length String`, `10` for `Packed Decimal`, and `2147483647` for `Binary`. The default value is 0 (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 110.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.
Signed	Check box	Specify whether the value is signed. This property is applicable only if the *Physical type* property is `Packed Decimal`. By default, this check box is cleared, which indicates that the value is not signed.
Justification	Enumerated type	If you have set the *Physical Type* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list. The option that you select determines the value that you must set for the property *Encoding Null Value*: • `NULLPadFill`. This option is valid only if *Physical Type* is `Fixed Length String`. The field is filled with the value specified by the *Padding Character*. The default value. • `NULLLogicalValue`. The *Encoding Null Value* property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This specifies that *Encoding Null Value* contains a value that is directly substituted as if it is a string. Use this option when the value you have set for *Encoding Null Value* to specify a null date is not a dateTime value, or does not conform to the standard dateTime format `yyyy-MM-dd 'T'HH:mm:ss`. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.
Encoding Null Value	String	If you set the *Encoding Null* property to `NULLPadFill`, this property is disabled. If you set the *Encoding Null* property to `NULLLogicalValue`, you must set this property to an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value conforming to `yyyy-MM-dd'T'HH:mm:ss` such as `1970-12-01`. If you set the *Encoding Null* property to `NULLLiteralValue`, you can enter any value that is the same length as the field. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`

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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for compound element decimal types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `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. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *Physical Type* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *Physical Type* 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.)
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a decimal element containing `1234` with a Virtual Decimal value of `3` is `1.234`, equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This option is valid only if *Physical Type* is `External Decimal`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for `NULLLiteralFill`. The default value is zero. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed list • Enter a character between quotation marks, for example `'c'` or `"c"`, where `c` is any alphanumeric character. • Enter a hexadecimal character 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for compound element float types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `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. • `Float`. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL and is the default value. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer`, `Packed Decimal`, and `Float` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Float`, select a value from the displayed list. The default value is 8. • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *Physical Type* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *Physical Type* 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.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.
Signed	Check box	Select or clear (unsigned, the default) this property. If you have set *Physical Type* to `Float`, this is selected. This property is used with *Sign Orientation*.
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).

Property	Type	Meaning
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a float element containing `1234` with a Virtual Decimal value of `3` is `1.234`. This property is not applicable if you have set *Physical Type* to `Float`.
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This option is valid only if *Physical Type* is `External Decimal`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for `NULLLiteralFill`. The default value is zero. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed list • Enter a character between quotation marks, for example `'c'` or `"c"`, where `c` is any alphanumeric character. • Enter a hexadecimal character 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for compound element integer types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `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. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you have set *Physical Type* to `Packed Decimal`, enter a value between 1 and 6. • If you have set *Physical Type* to `Extended Decimal`, enter a value between 1 and 11.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).

Property	Type	Meaning
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This option is valid only if *Physical Type* is `External Decimal`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for `NULLLiteralFill`. The default value is zero. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed list • Enter a character between quotation marks, for example `'c'` or `"c"`, where `c` is any alphanumeric character. • Enter a hexadecimal character 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for compound element string types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Compound elements

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element 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. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two 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. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. The default is Fixed Length String.
Length	Button and Integer	If you have selected a *Physical Type* of `Fixed Length String` or `Binary`, and have selected the length to be defined by *Length*, 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).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 110.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This is valid only if *Physical Type* is `Fixed Length String`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.
Encoding Null Value	STRING	The use of this property depends on the *Encoding Null* property. If specified, its length must be equal to the length of the string element, except for `NULLLiteralFill`. The default value is empty (not set). If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Compound element XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

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:

Field identification and physical representation.

The XML Wire Format properties described here apply to:

- Objects: Compound elements

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Physical representation

Property	Type	Meaning
Encoding	String	Select one of the following values from the drop-down list: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

XML wire format properties for compound element Boolean types:

Field identification.

The XML Wire Format properties described here apply to:

- Objects: Compound elements

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML wire format properties for compound element dateTime types:

Field identification and physical representation.

The XML Wire Format properties described here apply to:

- Objects: Compound elements

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

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 about the defaults for the dateTime format according to the logical type see “DateTime defaults by logical type” on page 790. See “DateTime formats” on page 782 for details of dateTime formats.

XML wire format properties for compound element decimal types:

Field identification.

The XML Wire Format properties described here apply to:

- Objects: Compound elements

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML wire format properties for compound element float types:

Field identification.

The XML Wire Format properties described here apply to:

- Objects: Compound elements

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML wire format properties for compound element integer types:

Field identification.

The XML Wire Format properties described here apply to:

- Objects: Compound elements

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

XML wire format properties for compound element string types:

Field identification.

The XML Wire Format properties described here apply to:

- Objects: Compound elements

Field identification

A number of the following properties 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: • `XMLElement`. 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 *XML Name*property to the same value, both objects must refer to the same element. This is the default value for element objects. • `XMLAttribute`. 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 *XML Name* property to different values. This is the default value for attribute objects. • `XMLElementAttrID`. 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 *ID Attribute Name* and a value as specified in *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrIDVal` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. • `XMLElementAttrVal`. 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 *Value Attribute Name*. • `XMLElementAttrIDVal`. This option combines the two options, `XMLElementAttrID` and `XMLElementAttrVal`. The object is rendered as a child of the parent complex type. The identity of the child is determined by the value of *ID Attribute Name*. The value is the value of *ID Attribute Value*. If you select this value for one object, and set this same value or the value `XMLElementAttrID` for a second object, and set *XML Name*, *ID Attribute Name*, *ID Attribute Value* to the same values: – You must also set *Value Attribute Name* to the same value for the two objects. – Both objects must refer to the same element. "XML rendering options" on page 752 shows some examples of how these rendering options affect the XML output, and provides usage recommendations.

Property	Type	Meaning
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 (_).
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is `id`.
Namespace	String	Enter the namespace associated with the *ID Attribute*.
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 *Render* is set to `XMLElement`, `XMLAttribute`, or `XMLElementAttrVal`. The default value is the identifier of the child.
Value Attribute Name	String	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 *Render*. The default value is `val`.
Namespace	String	Enter the namespace associated with the *Value Attribute*.

Compound element TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

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:

Field identification and physical representation.

The TDS properties described here apply to:

- Objects: Compound elements

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. The default is dependent on the setting of the message set property Derive default length from logical type. If Derive default length from logical type is selected, the default value is derived from any length or maxLength value constraint (schema facet) on the object's simple type.

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

TDS properties for compound element Boolean types:

Field identification and physical representation.

The TDS properties described here apply to:

- Objects: Compound elements

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Binary. The data is in bit string format. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

TDS properties for compound element dateTime types:

Field identification, physical representation, and null values.

The TDS properties described here apply to:
• Objects: Compound elements

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Length Encoded String 1. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a Length Encoded String 1 element is 255 length units. • Length Encoded String 2. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a Length Encoded String 2 element is 65535 length units. • Null Terminated String. The data string ends with the hexadecimal NULL character, `X'00'`. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • Binary. The data is in bit string format. • Time Seconds. This value supports C time_t, and Java Date and Time objects. It is valid only if the DateTime Format property represents numeric-only data. • Time Milliseconds. This value supports C time_t, and Java Date and Time objects. It is valid only if the DateTime Format property represents numeric-only data. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
DateTime Format	String	Specify a template for date and time. The default DateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 790. See “DateTime formats” on page 782 for details of date and time formats.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see "TDS Null handling options" on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

TDS properties for compound element decimal types:

Field identification, physical representation, numeric representation, and null values.

The TDS properties described here apply to:

• Objects: Compound elements

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • TLOG Specific - this option can be selected only if the Message Standard property of the message set is TLOG. This option indicates that the format of the data is specific to the TLOG messaging standard. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any minInclusive, maxInclusive, minExclusive, or maxExclusive value constraints (schema facets), on the simple type.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the Precision property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.
Precision	Button and Integer	This property is applicable only if Physical Type is Text. It is used if the value of the Virtual Decimal Point property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set Precision to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123. • If you set Precision 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 Precision to 2. • If you set Precision 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 Precision to 5. Or select one option from the list: • All Significant Digits - decimal separator only required if fractional digits (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • Explicit Decimal Separator - decimal separator always required: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present.
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.

Property	Type	Meaning
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

TDS properties for compound element float types:

Field identification, physical representation, numeric representation, and null values.

The TDS properties described here apply to:

• Objects: Compound elements

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Float. Equates to the data type FLOAT or DOUBLE in C, or the COMP-1 or COMP-2 numeric data type in COBOL. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified.

Property	Type	Meaning
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the Precision property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if Physical Type is Text. It is used if the value of the Virtual Decimal Point property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set Precision to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set Precision 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 Precision to 2. • If you set Precision 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 Precision to 5. Or select one option from the list: • All Significant Digits - decimal separator only required if fractional digits (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • Explicit Decimal Separator - decimal separator always required: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present. • Exponential Notation - Example "1.23456e002": data is written out to the bit stream as a signed value having the format `[sign1]a.bbbe[sign2]ccc` where: – `[sign1]` is the value of Negative Sign if the value is negative – `a` is a single decimal digit – `bbb` is one or more decimal digits – `[sign2]` is the value of Negative Sign if the exponent is negative – `ccc` is exactly three decimal digits (the exponent) `[sign1]` and `[sign2]` are absent if the value and exponent are positive. 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 that the value of Negative Sign is "-", and the value of Sign Orientation is `Leading`. The value -0.00012 is represented as `1.2*e*004` if Negative Sign is "*" and Sign Orientation is Trailing.
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.

Property	Type	Meaning
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

TDS properties for compound element integer types:

Field identification, physical representation, numeric representation, and null values.

The TDS properties described here apply to:

• Objects: Compound elements

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any minInclusive, maxInclusive, minExclusive, or maxExclusive value constraints (schema facets), on the simple type.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 770.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

TDS properties for compound element string types:

Field identification, physical representation, and null values.

The TDS properties described here apply to:

- Objects: Compound elements

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see "Regular expression syntax" on page 778.
Interpret Element Value	Enumerated type	Specify whether values stored within this object are interpreted as having significance for the parser and, if so, the type of interpretation that occurs. This interpretation is standard-specific and is therefore hard coded. The possible values for this property are: • None (the default value) • EDIFACT Service String • X12 Service String • Message Key • EDIFACT Syntax Level ID • HL7 Service String • HL7 Field Separator

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Length Encoded String 1. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a Length Encoded String 1 element is 255 length units. • Length Encoded String 2. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a Length Encoded String 2 element is 65535 length units. • Null Terminated String. The data string ends with the hexadecimal NULL character, `X'00'`. • TLOG Specific - this option can be selected only if the Message Standard property of the message set is TLOG. This option indicates that the format of the data is specific to the TLOG messaging standard. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see "TDS Null handling options" on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Embedded simple type properties

Logical, CWF, XML, TDS, and documentation properties for embedded simple types.

- "Embedded simple type logical properties" on page 619
- "Embedded simple type CWF properties" on page 620
- "Embedded simple type XML properties" on page 622
- "Embedded simple type TDS properties" on page 623
- "Documentation properties for all message set objects" on page 187

Embedded simple type logical properties:

The logical properties of an embedded simple type include properties that specify the number of occurrences of the embedded simple type.

Occurrences

Property	Type	Meaning
Min Occurs	Integer	Specify the minimum number of times that the object can repeat. The default value is 1. If the value is set to 0, the object is optional. With the exception of Max Occurs being set to -1, if a value is set for Min Occurs, it must be less than or equal to the value in Max Occurs.
Max Occurs	Integer	Specify the maximum number of times that the object can repeat. The default value is 1. If this property is not set, the object cannot occur more than once. If this property is set to 0, it is 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.

Embedded simple type CWF properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

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:

Physical representation, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

Physical representation

Property	Type	Meaning
Length	Button and Integer	If you have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1. The maximum value that you can specify is `2147483647`. The default value is empty (not set).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 110.
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for embedded simple type Boolean types:

Byte alignment and occurrences.

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: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for embedded simple type dateTime types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element 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. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two 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. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. • `Packed Decimal`. The dateTime is coded as a Packed Decimal number. It is valid only if the *DateTime Format* property represents numeric-only data. • `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 782 for details). • `Time Seconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *DateTime Format* property represents numeric-only data. • `Time Milliseconds`. This value supports C time_t and Java Date and Time objects. It is valid only if the *DateTime Format* property represents numeric-only data. The default value is `fixed length string`.
DateTime Format	String	Specify a template for date and time. The default dateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see "DateTime defaults by logical type" on page 790. See "DateTime formats" on page 782 for details of date and time formats.
Length	Button and Integer	If you have selected a *Physical Type* of `Fixed Length String`, `Packed Decimal`, or `Binary`, and have selected the length to be defined by *Length*, enter the number of length units for the element. The minimum value that you can specify is 1 for all three physical types. The maximum value that you can specify is `256` for `Fixed Length String`, `10` for `Packed Decimal`, and `2147483647` for `Binary`. The default value is 0 (zero).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see "Reordering objects" on page 110.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.
Signed	Check box	Specify whether the value is signed. This property is applicable only if the *Physical type* property is `Packed Decimal`. By default, this check box is cleared, which indicates that the value is not signed.
Justification	Enumerated type	If you have set the *Physical Type* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list. The option that you select determines the value that you must set for the property *Encoding Null Value*: • `NULLPadFill`. This option is valid only if *Physical Type* is `Fixed Length String`. The field is filled with the value specified by the *Padding Character*. The default value. • `NULLLogicalValue`. The *Encoding Null Value* property is first converted to an actual value, and rendered in the way specified for the field. • `NULLLiteralValue`. This specifies that *Encoding Null Value* contains a value that is directly substituted as if it is a string. Use this option when the value you have set for *Encoding Null Value* to specify a null date is not a dateTime value, or does not conform to the standard dateTime format `yyyy-MM-dd 'T'HH:mm:ss`. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.
Encoding Null Value	String	If you set the *Encoding Null* property to `NULLPadFill`, this property is disabled. If you set the *Encoding Null* property to `NULLLogicalValue`, you must set this property to an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value conforming to `yyyy-MM-dd'T'HH:mm:ss` such as `1970-12-01`. If you set the *Encoding Null* property to `NULLLiteralValue`, you can enter any value that is the same length as the field. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`

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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for embedded simple type decimal types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `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. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *Physical Type* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *Physical Type* 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.)
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.

Property	Type	Meaning
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a decimal element containing `1234` with a Virtual Decimal value of `3` is `1.234`, equivalent to 'V' or 'P' in a COBOL picture clause. There is no C equivalent
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This option is valid only if *Physical Type* is `External Decimal`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for `NULLLiteralFill`. The default value is zero. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed list • Enter a character between quotation marks, for example `'c'` or `"c"`, where `c` is any alphanumeric character. • Enter a hexadecimal character 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for embedded simple type float types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `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. • `Float`. This equates to the data type FLOAT or DOUBLE in C or the COMP-1 or COMP-2 data type in COBOL and is the default value. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer`, `Packed Decimal`, and `Float` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Float`, select a value from the displayed list. The default value is 8. • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you set the *Physical Type* to `Packed Decimal`, enter a value between 1 and 10. • If you set the *Physical Type* 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.)

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.
Signed	Check box	Select or clear (unsigned, the default) this property. If you have set *Physical Type* to `Float`, this is selected. This property is used with *Sign Orientation*.
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).

Property	Type	Meaning
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.
Virtual Decimal Point	Integer	Specify the number of places to the left (for a positive value) or right (for a negative value) that a decimal point is to be moved from its assumed position. For example, a float element containing `1234` with a Virtual Decimal value of `3` is `1.234`. This property is not applicable if you have set *Physical Type* to `Float`.
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This option is valid only if *Physical Type* is `External Decimal`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for `NULLLiteralFill`. The default value is zero. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed list • Enter a character between quotation marks, for example `'c'` or `"c"`, where `c` is any alphanumeric character. • Enter a hexadecimal character 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for embedded simple type integer types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `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. • `Packed Decimal`. Equates to the COMP-3 data type in COBOL. • `External Decimal`. Equates to the data type PIC 9 USAGE DISPLAY in COBOL. The representation of numeric elements can be affected by the Encoding and CodedCharSetId attributes that are set for the WebSphere MQ queue manager: • Elements that have *Physical Type* set to `Integer` and `Packed Decimal` are represented in the appropriate WebSphere MQ Encoding value. • Elements that have *Physical Type* set to `External Decimal` are represented in the WebSphere MQ CodedCharSetId value.
Length	Integer	Enter the number of bytes to specify the element length: • If you set the *Physical Type* to `Integer`, select `1`, `2`, or `4` (the default) from the displayed list. • If you have set *Physical Type* to `Packed Decimal`, enter a value between 1 and 6. • If you have set *Physical Type* to `Extended Decimal`, enter a value between 1 and 11.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count*. The default is `Bytes`.
Signed	Check box	Select (the default) or clear this property. This property is used with *Sign Orientation*.
Sign EBCDIC Custom Overpunched	Check box	If the *Physical Type* is set to `External Decimal` and the *Signed EBCDIC Custom* property is set, this indicates that the Sign EBCDIC Custom Overpunched representation is to be used within an ASCII environment. If this check box is not selected (the default), the Sign ASCII representation is used. The setting of the *Sign EBCDIC Custom Overpunched* check box is appropriate only if the *Sign Orientation* property is set to `Leading` or `Trailing` (indicating that the element/attribute has an embedded sign representation). The check box is not available if the element/attribute is unsigned (for example, if the *Signed* check box is not set).

Property	Type	Meaning
Sign Orientation	Enumerated type	If you have set *Physical Type* to `External Decimal` and you have selected *Signed*, choose from the following options that represent the COBOL options for displaying numeric data: • `Leading Overpunched`. This option sets a bit in the first byte 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'`. The default value. • `Leading Separate`. This option 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. • `Trailing Overpunched`. This option sets a bit in the last byte 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'`. • `Trailing Separate`. This option 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. If you have set *Physical Type* to any other value, the value `Not Applicable` is set for you.
Justification	Enumerated type	If you have set the *Physical Type* property to `External Decimal`, select `Left Justify` or `Right Justify` (the default value) from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	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 *Physical Type* property to `Extended Decimal`, and the *Justification* property is either `Left Justify` or `Right Justify`, specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, this is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This option is valid only if *Physical Type* is `External Decimal`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. You can specify a nonnumeric value for *Encoding Null Value*. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the *Encoding Null* property, except for `NULLLiteralFill`. The default value is zero. If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed list • Enter a character between quotation marks, for example `'c'` or `"c"`, where `c` is any alphanumeric character. • Enter a hexadecimal character 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

CWF properties for embedded simple type string types:

Physical representation, null values, byte alignment, and occurrences.

The Custom Wire Format properties described here apply to:

- Objects: Embedded simple types

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select one from the displayed list: • `Fixed Length String`. The element's length is determined by other length properties as follows. • `Length Encoded String 1`. The first byte of the element 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. • `Length Encoded String 2`. The element's first 2 bytes contain the length of the string following the two 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. • `Null Terminated String`. The string ends with the hexadecimal NULL character, `X'00'`. The default is Fixed Length String.
Length	Button and Integer	If you have selected a *Physical Type* of `Fixed Length String` or `Binary`, and have selected the length to be defined by *Length*, 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).
Length Reference	Button and Enumerated type	If you have selected the length to be defined by *Length Reference*, select the name of the integer object that specifies the length of this object. Make your selection from the displayed list of integer objects that are defined as siblings of the current object, and occur before it in the structure of the message. For information about reordering elements, see “Reordering objects” on page 110.

Property	Type	Meaning
Length Units	Enumerated type	Select the unit of length for the element or attribute. Select one of the following options from the displayed list (some physical types do not offer all these options): • `Bytes`. The length is given in bytes. • `Characters`. The Length is given in characters. This means that the number of bytes that are processed in the bitstream depends on the code page of the characters that are being processed. – For a single-byte code page (SBCS CCSID) such as "latin-1" (CCSID 850), the number of bytes is equal to the number of characters. – For a double-byte code page (DBCS CCSID) such as "UTF-16" (CCSID 1200), the number of bytes is exactly twice the number of characters. – For a multibyte code page (MBCS CCSID) such as "UTF-8" (CCSID 1208), the number of bytes depends on the bitstream content. The parser reads one character at a time and determines whether the character comprises one or more bytes. • `Character Units`. This option specifies that the size of each character (in bytes) is determined by the code page of the message. – For single-byte and double-byte code pages, this option is identical to `Characters`. – For a multibyte code page, this option provides improved parsing performance by assuming that every character is encoded in the smallest character unit that the code page supports. However, this means that a message must contain only these characters if it is to be processed correctly. For example, in code page "UTF-8" (CCSID 1208), the minimum character unit is 1 byte; therefore, the parser can make a single read (of the number of bytes specified by the `Length` property) to fetch the entire message. The message must contain only characters that are encoded in 1-byte units. • `End of Bitstream`. All data up to the end of the bitstream 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 for the *Length Count* or *Length Reference* property. The default is `Bytes`.
Justification	Enumerated type	If you have set the *Physical Type* property to `Fixed Length String`, select `Left Justify` (the default value) or `Right Justify` from the list. If you have selected another value for *Physical Type*, this property is inactive.

Property	Type	Meaning
Padding Character	String	If you have set the *Physical Type* property to `Fixed Length String`, and the *Justification* property is either `Left Justify` or `Right Justify`, this property is applicable. When writing an output message, use the padding character to fill out the remaining character positions when the string length is less than the length implied by the *Length* or *Length Reference* property. Whether the string is padded from the left or the right is governed by the *Justification* property. When parsing an input message, the padding character is trimmed from the end of the string. Whether the string is trimmed from the left or the right is governed by the *Justification* property. Specify this character in one of the following ways: • Select `NUL`, `'0'`, or `SPACE` from the displayed list. • Enter a character between quotation marks; for example, "c" or 'c', where c is any alphanumeric character. • Enter a Unicode value in the form `U+xxxx` where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. • Enter a hexadecimal character in the form 0xYY, where `YY` is a hexadecimal value. • Enter a decimal byte value (from 0 to 255). The choice of which of these padding character forms is used for an MRM element depends on the padding character that is required and whether the padding character is to be subject to data conversion. In most cases, the specification of a padding character in quotation marks is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is being generated. For example, when converting from ASCII to the code page 500, if you have specified U+0008 as your padding character, it is converted from 0x08 to 0x15, the ASCII and EBCDIC representations of 'back space'. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is being generated. If you are converting a message from one code page to another, ensure that the converted value of the padding character is valid for this code page. If the padding character cannot be represented in the target code page, it is replaced by a substitution character. The substitution character is fixed and its value depends on the specified target code page. If a padding character is required that is not subject to data conversion, the hexadecimal or decimal format can be used and you then have the option of specifying an absolute value as a padding character that is inserted directly into the output message. If this format is used, ensure still that this value is valid for the code page of any output messages that are created using these MRM definitions.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the displayed list: • `NULLPadFill`. This is valid only if *Physical Type* is `Fixed Length String`. The field is filled with the value specified by the *Padding Character*. *Encoding Null Value* must be set to an empty string. • `NULLLogicalValue`. The *Encoding Null Value* is transformed to match the required format for the field. The default value. • `NULLLiteralValue`. The *Encoding Null Value* is directly substituted as if it is a string. • `NULLLiteralFill`. The field is filled with the value specified by the *Encoding Null Value*. *Encoding Null Value* must resolve to a single character.
Encoding Null Value	STRING	The use of this property depends on the *Encoding Null* property. If specified, its length must be equal to the length of the string element, except for `NULLLiteralFill`. The default value is empty (not set). If you set the *Encoding Null* property to `NULLLiteralFill`, the value must resolve to a single character. Set the character in one of the following ways: • Select `SPACE`, `NUL`, `0x00` or `0xFF` from the displayed 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 format.

Byte alignment

Property	Type	Meaning
Byte Alignment	Enumerated type	Specify how the object is aligned from the start of the message. Select one of: • `1 Bytes`. The default value. • `2 Bytes` • `4 Bytes` • `8 Bytes` • `16 Bytes`
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.

Occurrences

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 displayed 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 no objects are listed, there are no integer objects before this one in the message structure. If a *Repeat Reference* is specified, it overrides any setting for the *Max Occurs* logical property when parsing and writing the message, but not for validation of the message.

Embedded simple type XML properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- ComIbmMrm _AnonBinary	- ComIbmMrm _AnonBoolean	- ComIbmMrm _AnonDate - ComIbmMrm _AnonDateTime - ComIbmMrm _AnonGDay - ComIbmMrm _AnonGMonth - ComIbmMrm _AnonGMonthDay - ComIbmMrm _AnonGYear - ComIbmMrm _AnonGYearMonth - ComIbmMrm _AnonTime	- ComIbmMrm _AnonDecimal
Float types	**Integer types**	**String types**	
- ComIbmMrm _AnonFloat	- ComIbmMrm _AnonInt	- ComIbmMrm _AnonString	

XML Wire Format properties for embedded simple type binary types:

Physical representation.

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: : • `CDatahex` (the default). Hexadecimal values in this field are specified with the CDATA qualifier, for example <e1><![CDATA[62]]></e1> • `hex`. Hexadecimal values in this field are specified as digits only, for example <e1>62</e1>. • `base64`. Values in this field are specified as digits only, coded in base 64.

XML wire format properties for embedded simple type Boolean types:

There are no properties to show.

The XML wire format properties described here apply to:
- Objects: Embedded simple types

XML wire format properties for embedded simple type dateTime types:

Physical representation.

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 about the defaults for the dateTime format according to the logical type see “DateTime defaults by logical type” on page 790. See “DateTime formats” on page 782 for details of dateTime formats.

XML wire format properties for embedded simple type decimal types:

There are no properties to show.

The XML Wire Format properties described here apply to:
- Objects: Embedded simple types

XML wire format properties for embedded simple type float types:

There are no properties to show.

XML wire format properties for embedded simple type integer types:

There are no properties to show.

XML wire format properties for embedded simple type string types:

There are no properties to show.

Embedded simple type TDS properties:

The properties, and their permissible values, vary according to the object type.

The properties that are 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 from those of type *Boolean*. Select the link for the object type from the following table.

Binary types	Boolean types	DateTime types	Decimal types
- ComIbmMrm _AnonBinary	- ComIbmMrm _AnonBoolean	- ComIbmMrm _AnonDate - ComIbmMrm _AnonDateTime - ComIbmMrm _AnonGDay - ComIbmMrm _AnonGMonth - ComIbmMrm _AnonGMonthDay - ComIbmMrm _AnonGYear - ComIbmMrm _AnonGYearMonth - ComIbmMrm _AnonTime	- ComIbmMrm _AnonDecimal
Float types	Integer types	String types	
- ComIbmMrm _AnonFloat	- ComIbmMrm _AnonInt	- ComIbmMrm _AnonString	

TDS properties for embedded simple type binary types:

Field identification and physical representation.

The TDS properties described here apply to:

- Objects: Embedded simple types

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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.

Property	Type	Meaning
Data Pattern	String	Specify the regular expression that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. The default is dependent on the setting of the message set property Derive default length from logical type. If Derive default length from logical type is selected, the default value is derived from any length or maxLength value constraint (schema facet) on the object's simple type.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

TDS properties for embedded simple type Boolean types:

Field identification and physical representation.

The TDS properties described here apply to:

- Objects: Embedded simple types

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Binary. The data is in bit string format. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

TDS properties for embedded simple type dateTime types:

Field identification, physical representation, and null values.

The TDS properties described here apply to:

- Objects: Embedded simple types

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Length Encoded String 1. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a Length Encoded String 1 element is 255 length units. • Length Encoded String 2. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a Length Encoded String 2 element is 65535 length units. • Null Terminated String. The data string ends with the hexadecimal NULL character, `X'00'`. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • Binary. The data is in bit string format. • Time Seconds. This value supports C time_t, and Java Date and Time objects. It is valid only if the DateTime Format property represents numeric-only data. • Time Milliseconds. This value supports C time_t, and Java Date and Time objects. It is valid only if the DateTime Format property represents numeric-only data. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
DateTime Format	String	Specify a template for date and time. The default DateTime format is dependent on the logical type of the object. For information about the defaults for the dateTime format according to the logical type, see “DateTime defaults by logical type” on page 790. See “DateTime formats” on page 782 for details of date and time formats.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see "TDS Null handling options" on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

TDS properties for embedded simple type decimal types:

Field identification, physical representation, numeric representation, and null values.

The TDS properties described here apply to:

• Objects: Embedded simple types

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • TLOG Specific - this option can be selected only if the Message Standard property of the message set is TLOG. This option indicates that the format of the data is specific to the TLOG messaging standard. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any minInclusive, maxInclusive, minExclusive, or maxExclusive value constraints (schema facets), on the simple type.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the Precision property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.
Precision	Button and Integer	This property is applicable only if Physical Type is Text. It is used if the value of the Virtual Decimal Point property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set Precision to 0, data is truncated so that the fractional part is lost. For example, the value 123.45 is truncated to 123. • If you set Precision 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 Precision to 2. • If you set Precision 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 Precision to 5. Or select one option from the list: • All Significant Digits - decimal separator only required if fractional digits (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • Explicit Decimal Separator - decimal separator always required: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present.
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.

Property	Type	Meaning
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

TDS properties for embedded simple type float types:

Field identification, physical representation, numeric representation, and null values.

The TDS properties described here apply to:

• Objects: Embedded simple types

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Float. Equates to the data type FLOAT or DOUBLE in C, or the COMP-1 or COMP-2 numeric data type in COBOL. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified.

Property	Type	Meaning
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.

Property	Type	Meaning
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Numeric representation

Property	Type	Meaning
Virtual Decimal Point	Button and Integer	Specify a non-zero integer that represents the position of an implied decimal point within a number, or specify 0, the default, to use the formatting of Float or Decimal numbers as specified by the Precision property. If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234 If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1,234,000.

Property	Type	Meaning
Precision	Button and Integer	This property is applicable only if Physical Type is Text. It is used if the value of the Virtual Decimal Point property is 0, which indicates that the decimal point is present in the data. It deals with truncation, and specifies how many digits are to follow the decimal point. Either specify a number of digits: • If you set Precision to 0, data is truncated so that the fractional part is lost. For example, the value `123.45` is truncated to `123`. • If you set Precision 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 Precision to 2. • If you set Precision 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 Precision to 5. Or select one option from the list: • All Significant Digits - decimal separator only required if fractional digits (the default): all significant digits are written to the output bit stream, and no decimal separator is written if no fractional digits are present. • Explicit Decimal Separator - decimal separator always required: all significant digits are written to the output bit stream and the decimal separator is always included, even when no fractional digits are present. The decimal separator must be present in the input bit stream, even when no fractional digits are present. • Exponential Notation - Example "1.23456e002": data is written out to the bit stream as a signed value having the format `[sign1]a.bbbe[sign2]ccc` where: – `[sign1]` is the value of Negative Sign if the value is negative – `a` is a single decimal digit – `bbb` is one or more decimal digits – `[sign2]` is the value of Negative Sign if the exponent is negative – `ccc` is exactly three decimal digits (the exponent) `[sign1]` and `[sign2]` are absent if the value and exponent are positive. 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 that the value of Negative Sign is "-", and the value of Sign Orientation is `Leading`. The value -0.00012 is represented as `1.2*e*004` if Negative Sign is "*" and Sign Orientation is Trailing.
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.

Property	Type	Meaning
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in “DateTime as string data” on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

TDS properties for embedded simple type integer types:

Field identification, physical representation, numeric representation, and null values.

The TDS properties described here apply to:

• Objects: Embedded simple types

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see “Regular expression syntax” on page 778.
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Integer. Equates to the data type SHORT or LONG in C, or the COMP, COMP-4, COMP-5, or BINARY numeric data type in COBOL. • Packed Decimal. The data is a packed decimal number that equates to the COMP-3 data type in COBOL. • External Decimal. The data is a decimal number that equates to the data type PIC 9 USAGE DISPLAY in COBOL. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.

Property	Type	Meaning
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any totalDigits value constraint (schema facet) or, if none, any minInclusive, maxInclusive, minExclusive, or maxExclusive value constraints (schema facets), on the simple type.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Numeric representation

Property	Type	Meaning
Sign Orientation	Enumerated type	The values that you can choose for this property are: • Leading Separate • Trailing Separate • Leading Overpunched • Trailing Overpunched This property is enabled only if you have set Physical Type to Text or External Decimal, and you have selected Signed. If Physical Type is Text, the only valid values of Sign Orientation are Leading Separate and Trailing Separate. If Physical Type is External Decimal and Sign EBCDIC Customer Overpunched is selected, the only valid values of Sign Orientation are Leading Overpunched and Trailing Overpunched.
Positive Sign	String	Specify the value that represents the positive symbol. Do not specify a numeric value. If no value is set, "+" is assumed. The positive sign is not written when creating an output message; it is used only to recognize the positive sign when parsing a message bit stream. This property is applicable only if Physical Type is Text and Signed is selected.
Negative Sign	String	Specify the value that represents the negative symbol. Do not specify a numeric value. If no value is set, "-" is assumed. This property is applicable only if Physical Type is Text and Signed is selected.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see “TDS Null handling options” on page 770.

Property	Type	Meaning
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

TDS properties for embedded simple type string types:

Field identification, physical representation, and null values.

The TDS properties described here apply to:

- Objects: Embedded simple types

Field identification

Property	Type	Meaning
Tag	String	Specify the value that is used to identify the object in a message bit stream. If the object is simple and the Data Element Separation 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. If the object is a complex element, and the Data Element Separation property of its parent is Tagged Delimited, Tagged Fixed Length, or Tagged Encoded Length, 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 that the parser uses to identify the data in the message to assign to the object. This property is used when the Data Element Separation method is set to Use Data Pattern in the complex type. For more details, see "Regular expression syntax" on page 778.
Interpret Element Value	Enumerated type	Specify whether values stored within this object are interpreted as having significance for the parser and, if so, the type of interpretation that occurs. This interpretation is standard-specific and is therefore hard coded. The possible values for this property are: • None (the default value) • EDIFACT Service String • X12 Service String • Message Key • EDIFACT Syntax Level ID • HL7 Service String • HL7 Field Separator

Property	Type	Meaning
Repeating Element Delimiter	String	Specify the delimiter to use between repeating elements. This delimiter is used only when the element repeats and the Data Element Separation property 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 previous conditions are true; if the messaging standard is HL7, the mnemonic <HL7_RS> is used; 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. If none of the previous conditions are true, a default is not applied.

Physical representation

Property	Type	Meaning
Physical Type	Enumerated type	Select the physical type of the object. If the Messaging Standard property of the message set is User Defined Text, User Defined Mixed, CSV, or TLOG, select one of the following values: • Text. The data is in character format. • Length Encoded String 1. The first byte of the data contains the length (in length units) of the data string that follows the length byte. The maximum length of a Length Encoded String 1 element is 255 length units. • Length Encoded String 2. The first two bytes of the data contain the length (in length units) of the data string that follows the two length bytes. The maximum length of a Length Encoded String 2 element is 65535 length units. • Null Terminated String. The data string ends with the hexadecimal NULL character, `X'00'`. • TLOG Specific - this option can be selected only if the Message Standard property of the message set is TLOG. This option indicates that the format of the data is specific to the TLOG messaging standard. The default is dependent on the Messaging Standard property. For all other Messaging Standard values, the Physical Type property is set to Text.
Length	Integer	Specify the expected length of the object in length units. A non-zero length must be specified if no Length Reference is specified. If this property is not set and the message set property Derive default length from logical type is selected, and the Physical type is 'Character', the default value is derived from any length or maxLength value constraint (schema facet) on the simple type.
Justification	Enumerated type	Specify the justification of the object if the data being written or parsed is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Select one of the following values from the list: • Not Applicable • Left Justify • Right Justify

Property	Type	Meaning
Padding Character	String	Specify the padding character to be inserted or interpreted on the writing or parsing of a fixed-length object, if the data is less than the fixed-length value. This property is used only when a value is written as a fixed-length string. Set this character in one of the following ways: • Select NUL, '0', or SPACE 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 Unicode value in the form `U+xxxx`, where `xxxx` is a Unicode value specified in hexadecimal. The maximum length of the string that you can enter is 10. The choice of which of these padding character forms is used for an MRM element depends on the padding character required and whether the padding character is subject to data conversion. In most cases, the specification of a padding character is sufficient, and when this padding character is used, it is converted to the target code page of the output MRM message that is generated. If a padding character is required that cannot easily be entered in the padding character field, the Unicode mnemonic format can be used to specify the required character. When used, this Unicode value is also converted to the target code page of the MRM message that is generated. If a padding character is required that is not subject to data conversion, the hexadecimal format can be used. This gives the option of specifying a padding character that is inserted directly into the output message. If this format is used, you must ensure that the hexadecimal value is valid for the code page of any output messages that are created using these MRM definitions. If you convert a message from one code page to another, you must ensure that the converted value of the padding character is valid for this code page. For example, when converting from ASCII to code page 500, if you have specified the numeric 8 as your padding character, it is converted from 0x08 to 0x15; the ASCII and EBCDIC representations of Backspace. There is a currently a restriction that the value of your padding character must not be greater than U+007F. If you enter a Unicode mnemonic or numeric value, it is considered to be the character that is represented by that number in UTF-8.
Length Reference	Enumerated type	This property is applicable only if Physical Type is Text, Binary, or TLOG Specific. If set, this property takes precedence over any value in the Length Units property. Specify the identifier of a sibling integer object, the value of which determines 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 110.

Representation of null values

Property	Type	Meaning
Encoding Null	Enumerated type	Select one of the following options from the list: • NULLPadFill. This option is valid only for fixed-length objects and is the default value. • NULLLogicalValue. The Encoding Null Value property is first converted to an actual value, and rendered in the way specified for the field. • NULLLiteralValue. This option specifies that Encoding Null Value 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 Encoding Null Value property to test or compare the content of the field in the message. • NULLLiteralFill. This option specifies that the field is filled with the value specified by the Encoding Null Value property. Encoding Null Value must resolve to a single character. The option that you select determines the value that you must set for the property Encoding Null Value. For full information about using these options, see "TDS Null handling options" on page 770.
Encoding Null Value	String	The use of this property depends on the Encoding Null property. The default value is zero. If you set the Encoding Null property for a dateTime object to NULLLogicalValue, the value that you set must be in an ISO8601 dateTime format. These formats are described in "DateTime as string data" on page 783. For example, specify a value that conforms to the yyyy-MM-dd'T'HH:mm:ss format; for example, `1970-12-01`.

Documentation properties for all message set objects:

Use the documentation property of an object to add information that describes the function of the object.

The documentation property is available on all objects except Key, Keyref, and Unique objects.

The property is a string field, and you can use any standard alphanumeric characters.

Additional MRM domain information

More information about the MRM domain.

MRM restrictions

The MRM parser does not exactly follow the XML Schema 1.0 specification.

However, the XMLNSC domain fully complies with the XML Schema 1.0 specification when validation is enabled. All of the constructs that are mentioned in this topic are supported by the XMLNSC domain.

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 the MRM domain.

- *Pattern facet on non-string data types*. The message broker only validates pattern facets that are applied to simple types based on xsd:string.
- *White space facet*. The message broker does not use the white space facet. However, if necessary, white space facets can be included in the message model. You can accurately control the processing of white space by 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 can be created and edited using the message editor, but the MRM domain processes them in a way that differs from the XML Schema specification.

- *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 or scenarios are not validated in exactly the same way as a validating XML parser would validate them:

- *Any Element or Any Attribute*. If the message model contains a wildcard ('any element' or 'any attribute'), the message broker validates the 'processContents' field as follows:
  - `skip`. No checking is done; any element or attribute is allowed.
  - `lax`. No checking is done; any element or attribute is allowed.
  - `strict`. Any element or attribute in the same message set is allowed.

  Note: If all of the definitions for a namespace are included within the same message set, the validation of 'strict' is the same as by a validating XML parser.
- *Element substitution and 'all' groups*. If an element can be substituted, and it occurs within an 'all' group, the following exceptions apply to the validation of the element:
  - The element is always validated as if it were optional.

– An input message is not rejected if more than one of the substitutions is used in the same 'all' group.

Data types for elements in an MRM message

A parser is supplied for the body of a message in the MRM domain; it associates each field with a specific data type.

The following table shows the mapping from XML Schema data types that you have specified for elements in the MRM to data types used by the broker and supported by ESQL. When you create an element, you might find that associated value constraints are created to ensure a more accurate mapping of the XML Schema type.

Data type of the element	ESQL data type in message tree
ANYURI	CHARACTER
BASE64BIN	BLOB
BOOLEAN	BOOLEAN
BYTE	INTEGER
DATE	DATE
DATETIME	TIMESTAMP
DECIMAL	DECIMAL
DOUBLE	FLOAT
DURATION	INTERVAL
ENTITIES	List of CHARACTER
ENTITY	STRING
FLOAT	FLOAT
GDAY	DATE
GMONTH	DATE
GMONTHDAY	DATE
GYEAR	DATE
GYEARMONTH	DATE
HEXBINARY	BLOB
ID	CHARACTER
IDREF	CHARACTER
IDREFS	List of CHARACTER
INT	INTEGER
INTEGER	DECIMAL
LANGUAGE	CHARACTER
LONG	INTEGER
NAME	CHARACTER
NCNAME	CHARACTER
NEGATIVE_INTEGER	DECIMAL
NMTOKEN	CHARACTER
NMTOKENS	List of CHARACTER
NON_NEGATIVE_INT	DECIMAL

Data type of the element	ESQL data type in message tree
NON_POSITIVE_INTEGER	DECIMAL
NORMAILIZED_STRING	CHARACTER
NOTATION	CHARACTER
POSITIVE_INTEGER	DECIMAL
QNAME	CHARACTER
SHORT	INTEGER
STRING	CHARACTER
TIME	DATETIME
TOKEN	CHARACTER
UNSIGNED_BYTE	INTEGER
UNSIGNEDINT	INTEGER
UNSIGNEDLONG	DECIMAL
UNSIGNED_SHORT	INTEGER

Simple type - list

In the message tree, a list type will be represented as a name node with an anonymous value child for each list item. This allows repeating lists to be handled without any loss of information. Repeating lists will appear as sibling name elements, each of which has its own anonymous value child nodes for its respective list items.

Additional CWF information

Information about data conversion and options for unll handling.

- "CWF data conversion"
- "CWF Null handling options" on page 750

CWF data conversion

You can convert an MRM message to a different code page or encoding, or both.

To do this, set the CodedCharSetId and Encoding fields in the Properties folder and the message tree to the target value.

The data conversion that is 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 binary, string, packed decimal, timeSeconds, or timeMilliseconds.

  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 (described below).

  If it is defined as a timeSeconds or timeMilliseconds value, it is converted as Integer (described below).

- Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong 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, 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, 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 310.

Additional XML information

Options for rendering, and null handling.

- "XML Null handling options"
- "XML rendering options" on page 752

XML Null handling options

The XML Wire Format supports the handling of null values in messages. Encoding null properties for XML are set only on the message set, and apply to all the defined objects in the message set.

You can use the following two properties to represent the numeric and non-numeric encoding for NULL in the XML Wire Format:
- Encoding Numeric Null
- Encoding Non-Numeric Null

These properties represent the numeric and non-numeric encoding for NULL respectively.

- The numeric data types are:
  - Decimal schema types: decimal, integer, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, unsignedLong
  - Float schema types: double, float
  - Integer schema types: byte, int, long, short, unsignedByte, unsignedInt, 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, ENTITIES, ENTITY, ID, IDREF, IDREFS, language, Name, NCName, NMTOKEN, NMTOKENS, normalizedString, NOTATION, QName, string, token

Each of these encodings has the following 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 causes 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>`
NULLValue	`zzz`	`<myElem>zzz</myElem>`
NULLXMLSchema		`<myElem xsi:nil='true'/>`[1] [5]
NULLValueAttribute		`<myElem></myElem>`[2] `<parent id="myElem"></parent>`[3]
NULLElement	null[4]	`<myElem><null/></myElem>`
NULLAttribute	null[4]	`<myElem null='true'/>`[1]

Notes:

1. The attribute must evaluate to true, so the value must be true, 1, or the Boolean True Value property.
2. This value is valid only for XMLElementAttrVal element rendering, as specified in “XML rendering options” on page 752. 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.
3. This value is valid only for XMLElementAttrIdVal element rendering, as specified in “XML rendering options” on page 752. 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.
4. Both NULLElement and NULLAttribute are deprecated. The element or attribute name provided must not include a namespace URI or prefix. If namespaces are enabled for the message set, the name matches any namespace.
5. `xsi:nil` is not supported with complex elements of MRM-XML.

XML Null value:

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. This behavior is in contrast to the TDS and CWF formats.

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.

Seting *Encoding Null Num* to `NULLEmpty` 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 with a binary object in XML, enter the appropriate 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 the message is written.

XML rendering options

You can use four properties on the XML layer that to affect how the XML messages are rendered.

The following table 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 should not 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 should not 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, you can use the Render property to achieve this.

Additional TDS information

More information about the TDS physical format.

- "TDS Industry standard formats"
- "Message characteristics" on page 762
- "TDS Null handling options" on page 770
- "TDS message model integrity" on page 772
- "Using regular expressions to parse data elements" on page 776

TDS Industry standard formats

WebSphere Message Broker supports the ACORD AL3, CSV, 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 172. 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 755
- "SWIFT messaging standard" on page 756
- "TLOG messaging standard" on page 757
- "X12 messaging standard" on page 757
- "ACORD AL3 messaging standard" on page 758
- "FIX messaging standard" on page 759
- "CSV messaging standard" on page 759

These topics also contain details of any predefined message set solutions that are available from IBM.

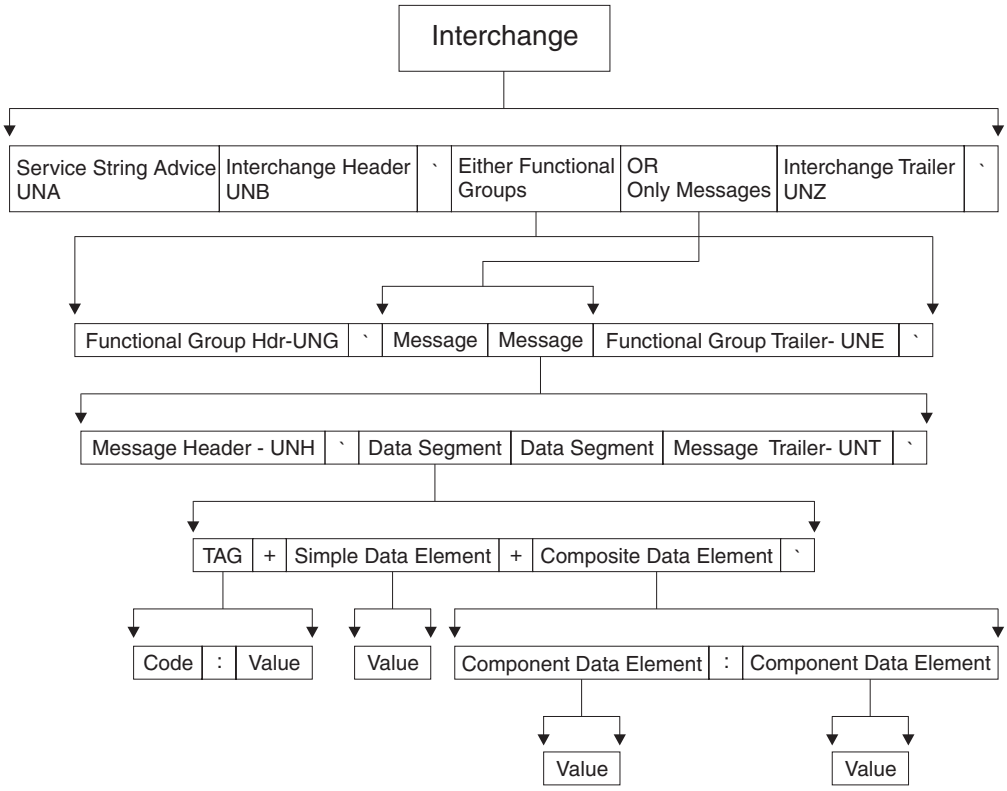
EDIFACT messaging standard:

EDIFACT, an international standard for EDI trading in commercial and non-commercial sectors, has an underlying syntax that 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 about 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 as follows:



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 predefined message set solution for EDIFACT can be purchased from IBM.

HL7 messaging standard:

The HL7 messaging standard defines the structure and content of messages that are exchanged between systems 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.

Predefined HL7 message sets are available as part of the following Healthcare sample:

- Healthcare

You can view samples only when you use the information center that is integrated with the Message Broker Toolkit.

SWIFT messaging standard:

SWIFT supplies secure, standardized messaging services and interface software to financial institutions.

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 following table.

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 predefined message set solution for SWIFT can be purchased from IBM. See the WebSphere MQ SupportPacs Web page.

Swift is a cooperative owned by the financial industry. For more information about the SWIFT messaging standard, see the SWIFT community Web site.

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 in character format or in a format that is specific to the message. This requires that the *Messaging Standard* property (at the message set level) is set to `TLOG`, and relevant objects that have this non-character based field in the TDS message have their *Physical Type* property set to `TLOG Specific`.

Predefined TLOG message sets are available as part of the following TLOG Processor samples:

- TLOG Processor

You can view samples only when you use the information center that is integrated with the Message Broker Toolkit.

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.

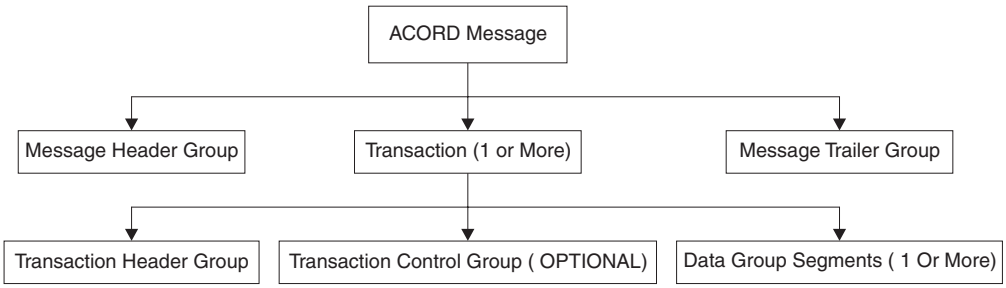
X12 messages can be modeled using the MRM Tagged/Delimited String Format (TDS).

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 predefined message set solution for X12 can be purchased from IBM.

ACORD AL3 messaging standard:

The basic structure of an ACORD AL3 message.



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 consist of fixed-length elements, therefore the type used can be modeled as:

```
Data Element Separation = Fixed Length
```

Two *Data Element Separation* methods are 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; 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 reversioning" on page 759

Using Fixed Length AL3:

Using Fixed Length AL3 is deprecated.

Fixed Length AL3 will be removed in a future version of the product; 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.

Different versions of the ACORD AL3 standard can be supported using the same message set. This value is like 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 reversioning:

Parse tags containing groups that are not in the current ACORD AL3 standards.

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 must also 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:

FIX messages can be modeled using the MRM Tagged/Delimited String Format (TDS).

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 about the FIX protocol, see the FIX protocol Web site.

A predefined message set solution for FIX can be purchased from IBM. See the WebSphere MQ SupportPacs Web page.

CSV messaging standard:

The comma separated value (CSV) format is a typical format for describing data in tables or spreadsheets.

The CSV format is used to exchange data between database applications or spreadsheet applications. Although the CSV format is widely used, a definitive specification has not been formally documented. However, these are some of the rules that characterize the CSV format:

- Data fields are separated by commas, and groups of data fields are separated by repeating field delimiters (for example, the <CR><LF> combination of ASCII characters).

  Here is a typical CSV message:

  ```
  12345,Smith,John,"3, North Street"<CR><LF>
  41352,Jones,Ivor,"5, South Road"<CR><LF>
  53421,Edwards,David,"10, East Lane"
  ```

- A comma that occurs within a data field is regarded as part of the data, rather than as a field separator, only if the comma is preceded by a special escape character (for example, a backslash (\)), or is surrounded by quotation marks (").

  For example, `Clapton, Eric, 461\, Ocean Boulevard, Scunthorpe` and `Clapton, Eric, "461, Ocean Boulevard", Scunthorpe` are equivalent; they both define data that contains four fields.

- A quotation mark character (") that is within a data field that is enclosed within quotation marks must always be 'escaped' by another instance of the quotation mark character.

  For example, `xx"xx` must be written as `"xx""xx"`, and `"xxxx"` must be written as `"""xxxx"""`.

- In an input message, any variable length data field can be enclosed within quotation mark characters, regardless of whether the field contains any special characters such as quotation mark characters, escape characters, or other reserved characters.

  The quotation mark characters must occur at the start and end of the data, are stripped from the data when the field is parsed, and are not added to the output tree. For example, the data `A,"B",C` results in an output tree that contains the values A, B, and C.

- If a data field contains two quotation mark characters and nothing else, the quotation mark characters are removed by the parser and the data field is processed in the same way as an empty field.

- In an output message, any data field that contains a quotation mark character, or any of the special characters that are specified in the TDS message set Reserved Characters property, has quotation mark characters added.

CSV messages can be modeled by using the MRM Tagged/Delimited String Format (TDS). The default message set property values are shown in “Default TDS message set properties” on page 172.

The following sample is a message set application that shows you how to model some typical CSV message variants, and how to transform the sample CSV messages to and from XML. The XML messages illustrate the logical structure of the data after it has been parsed.

- Comma Separated Value (CSV)

You can view samples only when you use the information center that is integrated with the Message Broker Toolkit.

You can also import a sample CSV message model by using the **New Message Definition File From IBM Supplied Message** wizard.

IDoc messaging standard:

Receiving data from SAP systems.

WebSphere Message Broker can receive data from SAP systems in various ways.

Two such ways are:

- ALE IDocs exported from SAP across the WebSphere MQ Link for R3.
- File IDocs exported from SAP to the file system.

Such IDocs are a fixed-length text format, and can be modeled using the MRM domain Tagged/Delimited String Format (TDS).

The IDOC domain is deprecated.

Note: For SAP data that is received from the WebSphere Adapter for SAP, use the DataObject domain.

Building the MRM TDS model for an IDoc:

The MRM domain Tagged/Delimited String (TDS) physical format is suitable for parsing and writing SAP ALE IDocs and SAP File IDocs. ALE IDoc messages are exported from SAP across the WebSphere MQ Link for R3. File IDocs are exported from SAP to the file system.

This topic describes how to build the message model that is required by the MRM parser when parsing and writing SAP ALE and File IDocs using its TDS physical format.

Obtaining the IDoc:

Create an import file of the required IDoc data for the Message Broker Toolkit.

1. Log on to an SAP system.
2. Run the supplied transaction `we60`, which extracts the IDoc data as a C header file.
   a. In **Basic Type**, select the IDoc type of interest; for example, `MATMAS02`.
   b. Leave the **Control**, **Data**, and **Status** check boxes cleared.
   c. Select the **Record types** version. A version 4 IDoc is type **3**.
   d. Press **F7** to display a C representation of the IDoc.
   e. Click **unconverted**.
   f. Select **System** → **List** → **Save** → **Local file**.
   g. When prompted, enter a file name and directory for the output from the transaction. The C representation of the IDoc is saved to this C header file.

Tip: The exported C header can be imported into the Message Broker Toolkit without any further manual processing. This situation was not true in previous releases of WebSphere Message Broker.

Modeling the IDoc:

Create your message model.

1. Switch to the Broker Application Development perspective of the Message Broker Toolkit.

2. Use the New Message Set wizard to create a message set for your IDoc. Select text data as the data to use. This action creates a Tagged/Delimited String Format (TDS) physical format, and presets the *Default message domain* property to MRM.
3. Use the Message Set editor to rename the TDS physical format to Text_IDoc.
4. Use the New Message Definition File From IBM supplied message wizard to import a prebuilt model of the overall ALE or File IDoc message structure. This model includes definitions of the DC and DD segments. The prebuilt models are called SAP ALE IDoc and SAP File IDoc. The resultant message definition file is called `ale_idoc.mxsd` or `file_idoc.mxsd`. For information about using the New Message Definition File From IBM supplied message wizard, see "Importing from IBM supplied messages" on page 135.
5. Use the New Message Definition File From C Header File wizard, or the mqsicreatemsgdefs command, to import the C representation of the IDoc into the new message set. Specify the following settings:
   - Set the Pre-processing option to SAP ALE IDoc or SAP File IDoc. If this option is not specified, the C header is not imported.
   - Create messages for the segments that appear in the IDoc.
   - Use the String Encoding option to import character arrays as fixed-length strings.
   - Use the Padding Char for String option to make space (" ") the padding character that is used.

   For information about using the wizard, see "Importing from C" on page 131.

Using the IDoc message model:

You can now use your message model to help you to construct a message flow that processes instances of your IDoc message, in the same way as any other message that belongs to the MRM domain.

Tip: SupportPac IA0F contains a more detailed description of the steps involved in building the IDoc message model. You can ignore utilities IDocHeaderTweak and IDocMsgSetTweak because that processing has been incorporated into the New Message Definition File From C Header File wizard.

Message characteristics

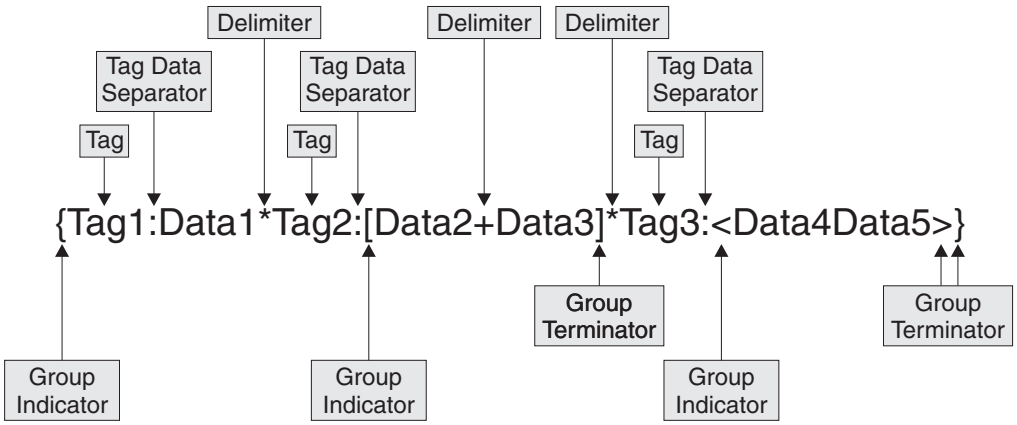
Features that are supported by the TDS wire format.

A number of features of text string messages 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 to 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 13. 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 764.
- 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 subfields. This is done by setting the regular expression in the `Data Pattern` property. See "Using regular expressions to parse data elements" on page 776 for further details.

The following figure 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 766

Specifying data element separation methods to model a message:

Specify the appropriate method of data element separation to identify data elements in a TDS 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 the 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 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:

```
tagA007dataAAAtagB006dataBBtagC009dataCCCCC
```

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):

```
tagA012dataAAAAAtagB010dataBBBtagC016dataCCCCCCCCC
```

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 778 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 26.

Specifying special characters to model a message:

You can specify a number of different types of special character in the workbench.

You can also specify special character values for message sets, types, and type members. The values that you set for a type override the corresponding values that are set for the message set in which it is defined.

You can specify a special character value in one of the following ways:

- As a literal string of one or more characters.

- As a mnemonic value.
- 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, complex type
Group Terminator	This is a string that indicates that the end of a group or complex type within a message	Message set, complex type
Tag Data Separator	This is the string that is used to separate a tag from its data.	Message set, complex type
Delimiter	This is the string used to separate data elements from one another	Message set, complex type
Repeating Element Delimiter	This is the string used to separate repeating data elements from one another	Local element or element reference
Tag	This is the string that indicates the start of a piece of data.	Local element or global element
Escape character	This is the character that is used to allow special reserved characters (such as delimiters) to be included as part of data	Message set
Quote character	This is the character that is used to allow special reserved characters (such as delimiters) to be included as part of data.	Message set
Reserved characters	These are characters that have a special meaning; for example, escape characters, quote characters, delimiters, and group indicators, are all examples of reserved characters.	Message set
Decimal point	This is the character that is used as the separator between the integer and fractional components of a decimal number.	Message set

If you create a complex type and set the *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 for special characters:

A mnemonic is a tag that is delimited by < and >. The broker translates the mnemonic to obtain the actual value of the special character.

Mnemonics can be used in TDS properties Decimal Point, Escape Character, Reserved Characters, Delimiter, Group Indicator, Tag data Separator, Tag, and Repeating Element Delimiter to specify special characters.

There are two types of mnemonic:

- Control code mnemonics, which map to the common non-printing characters.

  These are mapped using the local code page for your system. This is typically an ASCII code page on distributed platforms and an EBCDIC code page on other platforms.

  This means that characters are generally mapped to the 'expected' values for your system. This depends on your code page setting; for more information, refer to your system documentation. If a specific mnemonic is not mapped to the value that you need, you can use the explicit representation (<U+xxxx>, <0xNN>, or <0XNN>) that is 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' that is specified in the message itself.

Mnemonics can be specified in one of the following ways:

- `<Mnemonic_Name>`, where `Mnemonic_Name` can comprise alphanumeric characters and underscore (_) characters.
- `<U+xxxx>`, where `xxxx` are hexadecimal digits. The mnemonic is interpreted as the Unicode character that corresponds to the value of the digits.
- `<0xNN> or <0XNN>`, where `N` is a hexadecimal digit. The mnemonic is interpreted as the raw byte value given by the digits.

For more details about the supported mnemonics, see “TDS Mnemonics” on page 170.

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 following table.

Mnemonic string	Meaning	Default value	Associated property
<EDIFACT_CS>	Component separator in EDIFACT	:	Message set and complex type or group, *Delimiter*
<EDIFACT_DS>	Data element separator in EDIFACT	+	Message set and complex type or group, *Delimiter*

Mnemonic string	Meaning	Default value	Associated property
<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 or group, *Tag Data Separator*
<EDIFACT_DEC_NOTATION>	Decimal notation in EDIFACT	.	Message set, *Decimal Point*
<EDIFACT_ESC_CHAR>	Escape character in EDIFACT	?	Message set, *Escape Character*
<EDIFACT_GROUP_TERM>	Tag terminator in EDIFACT	'	Message set, *Group Terminator*
<X12_GROUP_TERM>	Tag terminator in X12	!	Message set level, *Group Terminator*
<X12_DS>	Data element separator for X12	*	Message set and complex type or group, *Delimiter*
<X12_CS>	Component separator for X12	:	Message set and complex type or group, *Delimiter*
<HL7_CS>	Component separator in HL7	^	Message set and complex type or group, *Delimiter*
<HL7_FS>	Data element separator in HL7	|	Message set and complex type or group, *Delimiter*
<HL7_RS>	Repeating element delimiter in HL7	~	Local element and element reference, *Repeating Element Delimiter*
<HL7_SCS>	Sub-component separator in HL7	&	Message set and complex type or group, *Delimiter*

Mnemonics for control characters are shown in the following table.

Mnemonic	Hex value	Unicode	Description
<ACK>	X'06'	<U+0006>	Acknowledge
<BEL>	X'07'	<U+0007>	Bell
<BS>	X'08'	<U+0008>	Backspace
<CAN>	X'18'	<U+0018>	Cancel
<CR>	X'0D'	<U+000D>	Carriage Return
<DC1>	X'11'	<U+0011>	Device Control One
<DC2>	X'12'	<U+0012>	Device Control Two
<DC3>	X'13'	<U+0013>	Device Control Three
<DC4>	X'14'	<U+0014>	Device Control Four
<DLE>	X'10'	<U+0010>	Data Link Escape
<EM>	X'19'	<U+0019>	End of Medium
<ENQ>	X'05'	<U+0005>	Inquiry
<EOT>	X'04'	<U+0004>	End of Transmission
<ESC>	X'1B'	<U+001B>	Escape
<ETB>	X'17'	<U+0017>	End of Transmission Block
<ETX>	X'03'	<U+0003>	End of Text
<FF>	X'0C'	<U+000C>	Form Feed

Mnemonic	Hex value	Unicode	Description
<FS>	X'1C'	<U+001C>	File Separator
<GS>	X'1D'	<U+001D>	Group Separator
<GT>	X'3E'	<U+003E>	Greater Than
<HT>	X'09'	<U+0009>	Horizontal Tabulation
<LF>	X'0A'	<U+000A>	Line Feed
<LT>	X'3C'	<U+003C>	Less Than
<NAK>	X'15'	<U+0015>	Negative Acknowledge
<NUL>	X'00'	<U+0000>	Null-
<RS>	X'1E'	<U+001E>	Record Separator
<SI>	X'0F'	<U+000F>	Locking Shift Zero (Shift In)
<SO>	X'0E'	<U+000E>	Locking Shift One (Shift Out)
<SOH>	X'01'	<U+0001>	Start of Heading
<SP>	X'20'	<U+0020>	Space
<STX>	X'02'	<U+0002>	Start of Text
<SUB>	X'1A'	<U+001A>	Substitute
<SYN>	X'16'	<U+0016>	Synchronous Idle
<US>	X'1F'	<U+001F>	Unit Separator
<VT>	X'0B'	<U+000B>	Vertical Tabulation

These mnemonics were created for characters that cannot be entered into the message editor.

You can enter a mnemonic in the form `<U+NNNN>`, where `NNNN` are hexadecimal digits. None of the characters in this structure are case-sensitive. Do not enclose spaces inside the angle brackets. These numbers represent a Unicode character, not a character in the code page of the input message.

You can enter a mnemonic in the form `<0xNN>`, where `NN` are hexadecimal digits. None of the characters in this structure are case-sensitive. Do not enclose spaces inside the angle brackets. These numbers represent a raw hexadecimal byte value, not a character in the code page of the input message.

If a mnemonic is of the form `<0xNN>`, it is applied directly to the input data, and no code page conversion takes place. Otherwise, a mnemonic is applied to the data after the data has been converted into Unicode from the code page of the input data.

TDS Null handling options

TDS supports the handling of null values within messages, provided that the logical Nillable property of the element is set.

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 enumerated values NULLPadFill, NULLLogicalValue, NULLLiteralValue, and NULLLiteralFill:

- Only use the NULLPadFill option 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; therefore, 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, set the message set level Trim Fix Len String property so that padded nulls are correctly parsed.
- If you select the NULLLiteralFill option, the value entered for the Encoding Null Value property is interpreted as a single character string value. Therefore, each character of the value of the element in the bit stream must match exactly the character value specified, to be interpreted as a null value.

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.

Handling missing fields in a delimited format

When dealing with delimited message formats, it is common for fields to be empty. For example, in a line-oriented format, blank lines might be inserted to separate lines of data.

```
This is Line 1<CR><LF>
<CR><LF>
This is Line 3<CR><LF>
This is Line 4
```

If the TDS property Suppress Absent Element Delimiters of the parent complex type is set to Never, such a message is successfully parsed, but the blank line does not appear in the message tree:

```
MRM
   - line1 = 'This is Line 1'
   - line3 = 'This is Line 3'
   - line4 = 'This is Line 4'
```

If you need to preserve the blank lines in the message tree, you can use TDS null handling to treat the blank line as NULL. You must set the following three properties on the element:

- Nillable = true
- TDS Encoding Null = 'NullLiteralValue'
- TDS Encoding Null Value = (Blank)

The message tree then looks like:

```
MRM
   - line1 = 'This is Line 1'
   - line2 = NULL
   - line3 = 'This is Line 3'
   - line4 = 'This is Line 4'
```

The example above assumes that each line is modeled as an element of simple type string. If each line is modeled as an element of complex type, with each line consisting of a repeating number of word elements, set the three null handling properties on the word element instead, because an element of complex type cannot have a null value.

The message tree then looks like:

```
MRM
   - line1
        - word = 'This'
        - word = 'is'
        - word = 'Line'
        - word = '1'
   - line2
        - word = NULL
   - line3
        - word = 'This'
        - word = 'is'
        - word = 'Line'
        - word = '3'
   - line4
        - word = 'This'
        - word = 'is'
        - word = 'Line'
        - word = '4'
```

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" on page 773
- "Restrictions for nesting complex types" on page 774
- "Omission and truncation of elements" on page 775

General rules: TDS message model integrity:

General rules govern 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 Length 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 778.

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 Length 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 does 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. Use element lengths.
- 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. Otherwise, *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 Length 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 following table.

	Parent				
Child	**Tagged Delimited, Tagged Encoded Length**	**All Elements Delimited, Variable Length 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 Length 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[1]	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:

Omitting and truncating elements depends on the setting of the property *Suppress Absent Element Delimiters*.

See "Complex type TDS properties" on page 234, "Global group TDS properties" on page 241, or "Local group TDS properties" on page 247 for a description of this property.

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 Length 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 ways provide techniques to avoid 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 Pn are the values of the elements of P and Cn 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

Use regular expressions to identify parts of an input message that are associated with subfields.

If your input messages can contain subfields 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 that conform 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 subfields. 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 with 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 778 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
"0x2A2B"	\x2A+	X'2A'
"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)
"ABCDEFGHIJ1234"	[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 that 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 Length 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 772 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 300 for details of valid settings of *Composition* and *Content Validation*.

Regular expression syntax:

Regular expression syntax elements and example syntax rules.

A regular expression allows you to specify the conditions that a string must satisfy. For example, you might use a regular expression to specify that a string must contain eight characters and start with an alphabetic character. Use the syntax in the following tables to write regular expressions to specify the sets of strings that are permitted. 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 that is supported is a subset of XML Schema regular expressions, with the addition of the \xNN hexadecimal syntax. 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:

Metacharacter	**Meaning**
\	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 [1]
[...]	match one of the class contained
[^...]	match one of the class not contained [1]
(...)	group the expressions [1]
|	match either preceding or following
Escape sequence	**Meaning**
\n	new line
\r	carriage return
\t	tab
\e	escape
Class code	**Meaning**
\d	digit [0-9]
\D	non-digit [^0-9] [2]
\s	white space[\t\n\r]
\S	non-whitespace character[^ \t\n\r] [2]
\p{L}	all letters [3]
\p{N}	all numbers, similar to \d [4]
[\p{N}\p{L}]	all numbers and all letters, similar to \w [4]
\P{L}	not letters, equivalent to [^\p{L}]
\P{N}	not numbers, equivalent to [^\p{N}]
\xNN	hexadecimal digits in the range 0 to F (\x00 not supported)
Range	**Meaning**
{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 776 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:

To parse messages in which fields are delimited by one of a set of characters or strings, set *Data Element Separation* to the method `Use Data Pattern`.

For example, consider a simple message with two numeric fields that can have either of the characters ';' or '/' delimiting them. You can use two approaches:

- 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 - 9, and the element `Delim` contains either ";" or "/".

- 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, where there is no markup to indicate the end of the repetitions.

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:

Take care when specifying regular expressions: some forms of regular expression can involve a large amount of work to find the best match, which might degrade 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 ".*;", which matches up to the *last* ';' character in the message, including all prior ';' characters in the matched text. Instead, use the pattern "[^;]*;".

Similarly, avoid using the pattern ".*", which always forces a search to the end of the message to try and find the best match, and therefore might result in poor performance. However, you must 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]+)*". Keep the expressions simple, with precise matching criteria. Simple expressions avoid the need to perform multiple searches for the best match.

DateTime formats

When you create an element or attribute with a simple type of dateTime, 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 XML, and all TDS and CWF physical types except those mentioned below. This is described further in “DateTime as string data” on page 783.
- Binary data. This is for the TDS or CWF Binary physical type. See “DateTime as BINARY data” on page 789 for more information.
- An offset from an epoch in seconds or milliseconds. This is used if you have set the TDS or CWF *Physical Type* property to `Time Seconds` or `Time Milliseconds` respectively. See “DateTime as encoded values” on page 790 for details of this option.

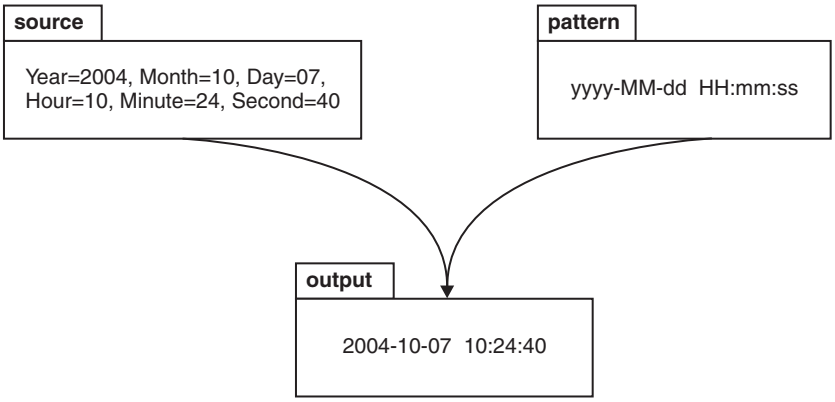
The defaults that are set for each message set property that relates to dateTime, for each physical representation (CWF, TDS, XML), are defined in “Message set defaults” on page 791.

DateTime as string data

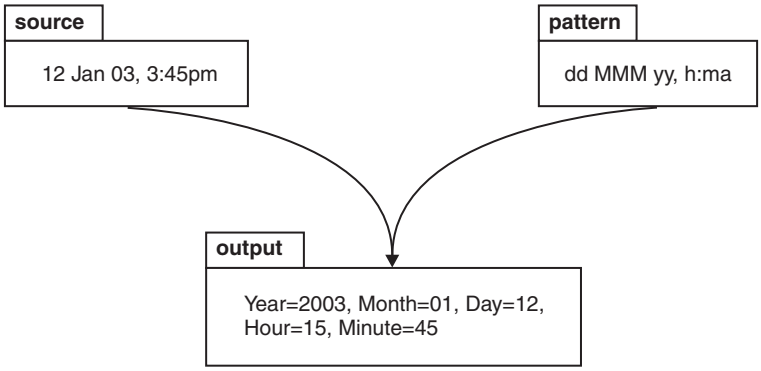
You can use a string of pattern letters to specify the dateTime format.

When you convert a date or time into a string, a format pattern must be applied that directs the conversion. Apply the format pattern to convert a date or time into a string, or to parse a string into a date or time.

During the conversion (for example, of a dateTime into a string), a pattern or a set of tokens is replaced with the equivalent source. The following diagram shows how a pattern is used to format a dateTime source to produce a character string output.



When a string is parsed (for example, when converting the string to a dateTime), the pattern or set of tokens is used to determine which part of the target dateTime is represented by which part of the string. The following diagram shows how this is done.



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 quotation mark is required within the string, use two single quotation marks ('').

Characters for formatting a dateTime as a string

The following table lists the characters that you can use in a pattern for formatting or parsing strings in relation to a dateTime. The table is followed by some notes that explain more about some of the examples in the table.

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)[1]	Number	2
EEE	day in week[1]	Text	Tue
EEEE	day in week[1]	Text	Tuesday
F	day of week in month (1-5)[2]	Number	2
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)[3]	Number	7
HH	hour of day in 24 hour form (00-23)[3]	Number	07
I	ISO8601 Date/Time (up to yyyy-MM-dd'T'HH:mm:ss.SSSZZZ)[4]	Text	2006-10-07T12:06:56.568+01:00
IU	ISO8601 Date/Time (similar to I, but ZZZ with output "Z" if the time zone is +00:00)[4]	Text	2006-10-07T12:06:56.568+01:00, 2003-12 -15T15:42:12.000Z
k	hour of day in 24 hour form (1-24)[3]	Number	8

Symbol	Meaning	Presentation	Examples
kk	hour of day in 24 hour form (01-24)[3]	Number	08
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
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[5]	Number	7
SS	centisecond[5]	Number	70
SSS	millisecond[5]	Number	700
SSSS	0.0001 second[5]	Number	7000
SSSSS	0.00001 second[5]	Number	70000
SSSSSS	0.000001 second[5]	Number	700000
T	ISO8601 Time (up to HH:mm:ss.SSSZZZ)[4]	Text	12:06:56.568+01:00
TU	ISO8601 Time (similar to T, but a time zone of +00:00 is replaced with 'Z')[4]	Text	12:06:56.568+01:00, 15:42:12.000Z
w	week in year[6]	Number	7, 53
ww	week in year[6]	Number	07, 53
W	week in month[7]	Number	2
yy	year[8]	Number	06
yyyy	year[8]	Number	2006
YY	year: use with week in year only[6]	Number	06
YYYY	year: use with week in year only[6]	Number	2006
zzz	time zone (abbreviated name)[9]	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

Symbol	Meaning	Presentation	Examples
'	escape for text		'User text'
''	(two single quotation marks) single quotation mark within escaped text		'o''clock'

The presentation of the dateTime object depends on the symbols that you specify.

- **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 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 is the upper bound for a particular symbol. For example, day in month has an upper bound of 31; therefore, a format string of `d` allows the values 2 or 21 to be parsed but does not allow the values 32 or 210 to be parsed. On output, numbers are padded with zeros to the specified length. A year is a special case; see note 8. Fractional seconds are also a special case; see note 5.
- **Lenient dateTime checking**: The parser converts out-of-band dateTime values to the appropriate in-band value. For example, the date 2005-05-32 is converted to 2005-06-01. Output of dateTimes always adheres 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 conversion allows dates such as 2006-01-123 and 2006-011-12, which are not valid, to be input. The value of 2006-01-123 is written as the date 2006-05-03, and the value of 2006-011-12 is written as the date 2006-11-12. The number of occurrences of the time zone formatting symbol Z applies only to the output dateTime format. White space is skipped over.
- **Physical Type**: If you specify the Physical Type property of the dateTime object to be Packed Decimal, the only pattern formatting symbols that are valid are those that represent numbers; that is, those that have Number in the Presentation column of the table. No other characters are allowed in the format pattern. For example, yyyyMMdd is valid, but yyyyMMMdd is not valid because MM is a numeric representation of the month, and MMM is a textual representation of the month.
- 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 are displayed in the resulting time text even if they are not enclosed within single quotation marks.
- You can create formatting strings that produce unpredictable results; therefore, 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 `3112006` might be interpreted as 3/11/2006 or as 31/1/2006.

Notes: The following notes apply to the preceding table.

1. The day in week field is the numeric offset into a week and varies according to the value of the physical message set property First Day of Week. For example, the third day in the week is Wednesday if the physical message set property First Day of Week is set to Monday.

2. 12th July 2006 is the second Wednesday in July and can be expressed as `2006 July Wednesday 2` using the format string `yyyy MMMM EEEE F`. Note that this format does not represent the Wednesday in week 2 of July 2006, which is 5th July 2006; the format string for this is `yyyy MMMM EEEE W`.
3. 24-hour fields might result in an ambiguous time, if specified with a conflicting am/pm field.
4. See "ISO8601, I and T DateTime tokens" on page 788.
5. Fractional seconds are represented by uppercase S. The length must implicitly match the number of format symbols on input. The format string `ss SSS` or `ss.SSS`, for example, represents seconds and milliseconds. However, the format string `ss.sss` represents a repeated field (of seconds); the value after the period (.) is taken as a seconds field, not as fractional seconds. The output is truncated to the specified length.
6. The start of a year typically 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; in this case, week 1 starts some days into the new year. Otherwise, the week is considered to be the first week of the new year; in this case, 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 lowercase `y` symbol, the adjustment is not done and unpredictable results might 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`

   Use `Y` only together 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 `2006-03-01` because the year input is ignored and the current year is assumed.
7. The first and last week in a month might include days from neighboring months. For example, Monday 31st July 2006 can be expressed as *Monday in week one of August 2006*, which is `2006 08 1 Monday` using format string `yyyy MM W EEEE`.
8. 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 `2006`, `yy` produces `06`.
   - 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.
9. Using the `zzz` option can have ambiguous results. For example, `BST` can be interpreted as `Bangladesh Standard Time` or `British Summer Time`. For compatibility reasons, WebSphere Message Broker uses the former interpretation.

To avoid these problems, use the zzzz option with a well-defined name; for example, Europe/London, Asia/Dhaka, or America/Los_Angeles.

ISO8601, I and T DateTime tokens

If your dateTime values comply with the ISO8601:2000 'Representation of dates and times' standard, consider using the formatting symbols I and T, which match the following subset of the ISO8601 standard.

- 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 (subsections c, e, and f)
  - Extended format (subsections a, b, and d)

Use the formatting symbols I and T only on their own:

- The I formatting symbol matches any dateTime string that conforms to the supported subset.
- The T formatting symbol matches any dateTime string that conforms to the supported subset that consists of a time portion only.

The following table shows how the output form relates to the logical data type.

Logical model data type	ESQL data type	Output form
xsd:dateTime	TIMESTAMP or 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 both '+00:00' and 'Z' to indicate a zero time difference from Coordinated Universal Time (UTC), but on output they always generate '+00:00'. If you want 'Z' to always be generated on output, use the IU or TU formatting symbols instead.
- ZZZ always writes '+00:00' to indicate a zero time difference from Coordinated Universal Time (UTC). If you want 'Z' to always be generated on output, use ZZZU 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 Coordinated Universal Time (UTC) by using either the Z symbol 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 is displayed by using the Default DateTime Format property of the element or attribute. Alternatively, you can preserve the input UTC format by selecting the message set property Use input UTC format on output. If this property is selected,

the UTC format is preserved in the output message and overrides the format that is 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"	2006.07.10 at 15:08:56 -05:00
"EEE, MMM d, "yy"	Wed, July 10, '06
"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.MMMMM.dd hh:mm aaa"	1996.July.10 12:08 PM

DateTime as 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 be used only in groups of 1, 2, or 4; for example, y, yy, or yyyy.

The following table shows the dateTime symbols for 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 saving time flag */
};
```

You can format this structure by specifying the string "`ssssmmmmHHHHddddMMMM+1yyyy+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, `yyyy+1900` adds 1900 to the year. `X` expects one byte of input, but ignores its value. On output, it writes the byte as 0.

DateTime as encoded values

You can represent a dateTime element with the `TimeSeconds` and `TimeMilliSeconds` physical types.

`TimeSeconds`
A 4 byte integer that represents the number of seconds since the epoch.

`TimeMilliSeconds`
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 a format string. To change the epoch you must update the physical properties of your dateTime element:

- In the Physical representation section, you must set the Physical Type to either Time Seconds or Time Milliseconds.
- In the Format field, set the value to the format of "yyyy-MM-dd'T'HH:mm ZZZ". For example, 2000-01-01T12:59 +00:00.

DateTime defaults by logical type

The default value that is assigned to the dateTime Format property is dependent on the logical type of the property.

The following table lists the default 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 element 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

Component	Default value
Hour	First hour of day
Minute	Minute 0 of hour
Second	Second 0 of minute
Millisecond	Millisecond 0 of second

Message set defaults

The default dateTime formatting property settings for the different MRM physical formats.

Message set property	CWF default	TDS default	XML default
Default DateTime Format	See Note 1.	See Note 1.	See Note 1.
Default Time Zone ID	Use Broker Locale (see Note 2)	Use Broker Locale (see Note 2)	Use Broker Locale (see Note 2)
Century Window	53	53 (80 for SWIFT)	53
Days in First Week of Year	4	Use Broker Locale (see Note 2)	Use Broker Locale (see Note 2)
First Day of Week	Monday	Use Broker Locale (see Note 2)	Use Broker Locale (see Note 2)

Note:

1. You can either set the default dateTime format to be derived from its logical type (the default), or specify the dateTime format that is 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 156, "TDS Format message set properties" on page 162, or "XML Wire Format message set properties" on page 178.

Additional MIME domain information

Standard header fields, and parser use and restrictions.

- "MIME standard header fields"
- "MIME parser use and restrictions" on page 795

MIME standard header fields

Check this quick reference to the common MIME headers.

This information does not provide a definitive specification of MIME. In some cases, the MIME parser allows documents that 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 takes special note only of the Content-Type header field.

All MIME headers can 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/plain`. Content-Type defines the type of data in each part as a type/subtype. The MIME parser accepts most values for Content-Type and 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 required for each nesting level.

Syntax:

```
Content-Type: type/subtype;parameter
```

where type and subtype define the Content-Type, and all 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`). Although this structure is syntactically correct, because a valid boundary parameter does not exist, this message is 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 optionally can be enclosed in quotation marks. When it appears in the MIME body, the value is preceded by the sequence '--', and you must ensure that the resulting value (in this example, --Boundary) cannot appear in the message body. If the message data is encoded as quoted-printable, you must include a boundary that includes a sequence such as “=_”, which cannot appear in a quoted-printable body.

Some common Content-Type values are shown below. Other values are allowed, and stored in the logical tree.

Content-Type	Description
text/plain	Typically used for a typical mail or news message. text/richtext is also common.
text/xml	Typically 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 with the characters 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, and can include XML, which typically uses UTF-8 or UTF-16 encoding. This type of data cannot be transmitted over some transport protocols, and might 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 following values only are allowed by the WS-I Basic Profile:

- 7bit - the default
- 8bit
- binary
- base64
- quoted-printable

The values 7bit, 8bit, and binary all effectively mean that no encoding took place. 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 field 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

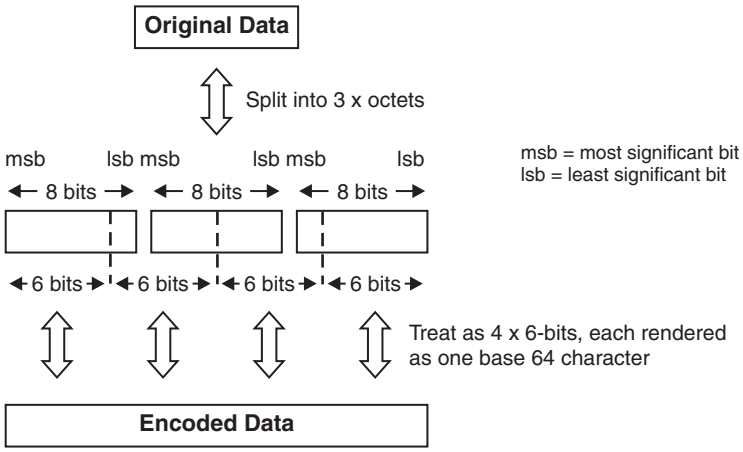
Optional. This field enables parts to be described.

MIME encodings

The following section provides a basic guide to the base64 and quoted-printable encoding; refer to RFC 1521 (linked at the end of this topic) 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).



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 all 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 appropriate only if most of the data comprises printable characters. Specifically, characters in the ranges 33-60 and 62-126 are typically represented by the corresponding ASCII characters. Control characters and 8-bit data must be represented by the sequence = followed by a pair of hexadecimal 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 hexadecimal 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.

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

The MIME domain does not support the full MIME standard, but supports specific known uses of MIME. Read this general introduction to the MIME parser, and information about some of the restrictions in its use.

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 you to parse and write multipart MIME messages.

MIME is used to send e-mail messages. When the e-mail includes attachments, 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 that consist of multiple separate parts.

Examples are:

- **RosettaNet.** Each part is typically a separate XML document but there might also be non-XML attachments. 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).** 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 in which 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, the record can either be processed before it is uploaded or it can be sent unchanged.

Restrictions

The MIME parser is driven by bit streams 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. You can parse specific MIME parts by using a different parser. If you are using an MRM parser, messages can be validated in the usual way. The MIME parser does not support on-demand parsing and ignores the Parse Timing property.

You can specify the new MIME domain either at run time in an MQRFH2 header (WebSphere MQ only) or statically in their message flow in the tooling (on the input nodes MQGet, HTTPRequest, ResetContentDescriptor, or XSLTransform). The MIME parser is then 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 bit streams. 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, set the Content-Type value using the ContentType property in the MIME domain. Setting

the Content-Type value directly in the MIME tree or HTTP trees can lead to the value being ignored or used inconsistently.

Typically, the MIME parser handles the majority of uses of MIME in application-to-application messaging, including multipart MIME with a single part and non-multipart MIME documents. However, you should use the SOAP domain for SOAP with Attachments (SwA).

Additional IDOC domain information

More information about the deprecated IDOC domain.

This section provides additional information in relation to the IDOC domain. This information is categorized into:

- "Building the message model for the IDOC parser"
- "Field names of the IDOC parser structures" on page 798

Note: The IDOC domain is deprecated and is not recommended for developing new message flows. Instead use the MRM domain with a TDS physical format when you want to process SAP ALE IDocs that are sent to the broker by SAP R3 clients across the WebSphere MQ link for R3.

Building the message model for the IDOC parser

The ALE IDoc messages that are sent to, and received from, SAP applications by using the WebSphere MQ Link for R3, can be processed by the IDOC parser, which requires a message model to interpret the data correctly. This topic describes how to build the message model.

The IDOC domain is deprecated. To develop new message flows, use the MRM domain with a TDS physical format when you want to process SAP ALE IDocs that are sent to the broker by SAP R3 clients across the WebSphere MQ link for R3.

Obtaining the IDoc

Create an import file of the required IDoc data for the Message Broker Toolkit.

1. Log on to an SAP system.
2. Run the supplied transaction `we60`, which extracts the IDoc data as a C header file.
   a. In **Basic Type**, select the IDoc type of interest; for example, `MATMAS02`.
   b. Leave the **Control**, **Data**, and **Status** check boxes cleared.
   c. Select the **Record types** version. A version 4 IDoc is type `3`.
   d. Press **F7** to display a C representation of the IDoc.
   e. Select **System** → **List** → **Save** → **Local file**.
   f. Click **unconverted**.
   g. When prompted, enter a file name and directory for the output from the transaction. The C representation of the IDoc is saved to this C header file.

Tip: The exported C header can be imported into the Message Broker Toolkit without any further manual processing.

Modeling the IDoc

Create your message model.

1. Switch to the Broker Application Development perspective.
2. Use the New Message Set wizard to create a message set for your IDoc. Select binary data as the data to use. This option creates a message set with a Custom Wire Format (CWF) physical format, and presets the Default message domain property to MRM.
3. Use the Message Set editor to change the Default message domain property to IDOC.
4. Use the New Message Definition File wizard to import a prebuilt model of the ALE IDoc message structure. To start the wizard, click **File** → **New** → **Message Definition File From**. When the wizard opens, select IBM supplied message, then SAP ALE IDoc. This SAP ALE IDoc prebuilt model includes definitions of the DC and DD segments. The resulting message definition file is called `ale_idoc.mxsd`. For information about using the New Message Definition File wizard, see "Importing from IBM supplied messages" on page 135.
5. Use the New Message Definition File wizard, or the mqsicreatemsgdefs command, to import the C representation of the IDoc into the new message set. To start the wizard, click **File** → **New** → **Message Definition File From**.

   Specify the following settings:

   - Select C Header file.
   - Set Select the pre-processing option to apply to SAP ALE IDoc. If this option is not specified, the import of the C header fails. If this option is specified, the message prefix preference is ignored.
   - Create messages for the segments that appear in the IDoc.
   - Use the String Encoding option to import character arrays as fixed-length strings.
   - Use the Padding Char for String option to make space (" ") the padding character that is used.

   For information about using the New Message Definition File From C Header File wizard, see "Importing from C" on page 131.

Using the IDoc message model

You can now use your message model to help you to construct a message flow that processes instances of your IDoc message. You can use ESQL or Java to access the fields of the IDoc. You cannot use graphical maps to access the fields of the IDoc because the IDOC domain is not supported by the mapping editor.

When you set the properties of the MQInput node that is to receive your IDoc from the WebSphere MQ Link for R3, the Message Domain property must be IDOC, the Message Set property must be the name of your message set, and the Message Format property must be the name of your Custom Wire Format. You do not need to set a Message Type property on the MQInput node because it is not needed by the IDOC parser.

When your message flow is complete, add the message set and the message flow to a broker archive (BAR) file and deploy the BAR file to a broker execution group.

When an IDoc is received by the MQInput node, the IDOC parser processes the SAP-defined elements in the DC, then, for each DD, processes the SAP-defined elements, and calls the MRM parser to process the user-defined segment data, as

described by your exported IDoc, using the CWF physical format. The MRM parser knows the Message Type property to use for the user-defined segment, because it is obtained from the SAP-defined DD field *segnam* by the IDOC parser.

Tip: SupportPac IA0F contains a more detailed description of the steps involved in building the IDoc message model. You can ignore utilities IDocHeaderTweak and IDocMsgSetTweak because that processing has been incorporated into the New Message Definition File From C Header File wizard.

Field names of the IDOC parser structures

The field names of the Control Structure (DC) and the Data Structure (DD) that are used by the IDOC parser.

The field names are documented in the form that they are used in a SET statement of ESQL; for example:

```
SET OutputRoot.Properties = InputRoot.Properties;
SET OutputRoot.MQMD = InputRoot.MQMD;
```

Control structure (DC) fields

All the fields must be specified and set.

The syntax is:

```
<rootname>.<ParserName>.<foldername>.<fieldname>=
```

For example:

```
SET "OutputRoot"."IDOC"."DC"."docnum" = '0000000000000001';
SET "OutputRoot"."IDOC"."DC"."idoctyp" = 'MATMAS01'
```

The field names, which must be specified in order, are:

1) tabnam	2) mandt	3) docnum
4) docrel	5) status	6) direct
7) outmod	8) exprss	9) test
10) idoctyp	11) cimtyp	12) mestyp
13) mescod	14) mesfct	15) std
16) stdvrs	17) stdmes	18) sndpor
19) sndprt	20) sndpfc	21) sndprn
22) sndsad	23) sndlad	24) rcvpor
25) rcvprt	26) rcvpfc	27) rcvprn
28) rcvsad	29) rcvlad	30) credat
31) cretim	32) refint	33) refgrp
34) refmes	35) arckey	36) serial

Data structure (DD) fields

To access each DD segment, use the array suffix DD[1], DD[2], and so on.

The syntax is:

```
<rootname>.<ParserName>.DD[1].<fieldname>=
```

For example:

```
SET OutputRoot.IDOC.DD[I].segnam  = 'E2MAKTM001';
SET OutputRoot.IDOC.DD[I].mandt2   = '111';
```

The following table illustrates how the suffix 2 is used to give unique field names to the `mandt` and `docnum` fields.

The field names, which must be supplied in order, are:

1) segnam	2) mandt2	3) docnum2
4) segnum	5) psgnum	6) hlevel

Notes:

- The last 1000 bytes of data in the DD segment are the bytes that are parsed by the MRM domain.
- The DD `segnam` describes the model that the MRM uses.

Segment fields

The syntax is:

```
<rootname>.<ParserName>.DD[1].sdatatag.MRM.<fieldname>=
```

For example:

```
SET OutputRoot.IDOC.DD[I].sdatatag.MRM.msgfn = '006'
SET OutputRoot.IDOC.DD[I].sdatatag.MRM.spras_iso = 'EN'
```

Notes:

- The `sdatatag` field indicates to the parser that it is the element that contains the data to be manipulated.
- The `MRM` field indicates that the MRM handles the transformation.

msgfn	spras	maktx
msgfn	spras_iso	fill954

The `fill954` field is the filler for the segment because an incoming IDoc to SAP must have 1000 byte segments.

Message model task list errors that have a quick fix

You can apply a quick fix to some message modeling task list warnings or errors 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

Error type	Description	Quick Fix
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. (You should use this if you only have 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. You should use this if you only have Version 6.0 brokers in your domain.)

Error type	Description	Quick Fix
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: • Will update your definition to unset the "Repeat Count" property. • Will update your definition to set "Max Occurs" to the value of the "Repeat Count" property, and to unset the "Repeat Count" property.
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: • Will update your simple type definition so that the Length facet is converted to a Max length facet. • Will update all the simple type definitions in your message definition file so that all Length facets are converted to Max Length facets, then save the file to remove all the associated task list errors.
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 invalid facets and all duplicate facets are removed.

Generated model representations

Information about generating documents, WSDLs, and XML schemas.

Document generation

The document generator produces a set of HTML pages and any necessary files (for example, images) that are required to display the pages correctly.

Output Files

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 1. WSDL File Naming Convention

File	File Name	File Extension	Value of name attribute on WSDL <definitions> element
Service File (single-file format)	<Message Set>	wsdl	<Definition Name>
Service File (multi-file format)	<Message Set>Service	wsdl	<Definition Name>Service
Binding File	<Message Set>Binding	wsdl	<Definition Name>Binding
Interface File	<Message Set>	wsdl	<Definition Name>

If '`Deployable WSDL`' is generated, no additional XML schema (xsd) files are generated, and the WSDL refers directly to the broker message definition (mxsd) files; otherwise, separate XML schema (xsd) files are generated, unless you selected '`inline schema`'.

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 – for example, portType / operation.

The following values apply to the WSDL definition as a whole:

Table 2. WSDL objects

Element	Attribute	Value
definitions	xmlns	assign namespace prefixes.
definitions	targetNamespace	This is the WSDL Namespace solicited by the wizard, defaulting to http://tempuri.org/<Message Set>.
message	name	<operation>_<role> where <operation> is the operation name and <role> is in, out, or fault
part	name	name of the broker message. 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>PortType
binding	name	• "<Message Set>SOAP_HTTP_ Binding" • "<Message Set>SOAP_JMS_ Binding"
soap:binding	style	From the value of Style set in the wizard.

The following values apply to each individual WSDL operation:

Table 3. WSDL <operation> objects

Element	Attribute	Value
operation	name	The name of the operation specified in the wizard.
soap:operation	style	From the value of Style set in the wizard.
input, output, fault	name	<operation>_<role>, where <operation> is the operation name, and <role> is Input, Output, or Fault.
soap:body	namespace	• If Style has been set to rpc, it is the namespace of the corresponding broker message. • If Style has been set to document the attribute is not generated.
soap:header, soap:fault, soap:body	use	This is set to *literal*.

Message Set

The message set provides the basis for many important broker features, including mapping support and ESQL code completion at development time, and validation at run time.

Therefore, the WSDL that you use in the broker at development time (for example, when configuring SOAP nodes) is integrated with the message set, and references the broker message definitions (mxsd) rather than ordinary Schema (xsd) files. This is referred to as *deployable WSDL* and is displayed under the category `Deployable WSDL` in the workbench.

Deployable WSDL is generated when you specify your Message Set Folder (the immediate child of your Message Set Project) as the `destination directory` for your WSDL.

Otherwise, regular WSDL is generated, along with separate XML schema (xsd) files if these were requested. Regular WSDL cannot be used to configure SOAP nodes, but is suitable for consumption by external applications such as .NET.

Assuming that you are generating deployable WSDL for use in a message flow, the flow typically needs to be able to parse and validate the runtime SOAP messages described by that WSDL. The WSDL generator, therefore, adds additional definitions to your message set:

- For rpc-style WSDL, additional definitions for the WSDL operations themselves are added to your message set
- For the version of the SOAP Envelope used by the WSDL an mxsd file is added – this will be soapenv11.mxsd or soapenv12.mxsd.
- For use by ESQL Content assist and the Mapping editor primarily, a definition of the SOAP_Domain_Msg tree.

XML Schema generation

The behavior of XML Schema generation. For example, use the schema generated from a message definition file to 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 752 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 deprecated objects that are only encountered if the message set was created using WebSphere MQ Integrator Broker Version 2.1.

They 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

Supported features of formats that have been imported from C, COBOL, SCA Export, SCA Import, WSDL, or XML Schema.

This section provides information about the supported features of formats that have been imported from an external source. Details are provided for:

- "Importing from C: supported features" on page 807
- "Importing from COBOL: supported features" on page 809
- "Importing from WSDL: generated objects and restrictions" on page 814

- "Importing from XML Schema: unsupported features" on page 817

Importing from C: supported features

The C importer uses default values when mapping C data types to message model elements.

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 character indicates that it is an anonymous simple type based on this type. For strings, the purpose of the anonymous type is to add a maximum length restriction; for numeric types, the purpose of the anonymous type is to add either a minimum or a maximum value restriction.

C data type	XML Schema data	Notes
Char	xsd:string-	maxlength=1
Char[10]	xsd:string-	maxlength=10
Char[10][3]	xsd:string-	maxlength=3
Char[10][3][6]	xsd:string-	maxlength=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 (see note 1)	xsd:double	
<any pointer type>	xsd:hexBinary-	maxlength=(see note 2)
<any enum>		(see note 3)

The following table shows how C definitions influence the physical MRM CWF characteristics of the elements that are generated in the message model.

C data type	CWF Physical type	CWF Length characteristics	Other CWF characteristics
Char	Fixed Length	Length = 1 Length Units = Bytes	
Char[10]	Fixed Length	Length = 10 Length Units = Bytes	Left justify

C data type	CWF Physical type	CWF Length characteristics	Other CWF characteristics
Char[10][3]	Fixed Length	Length = 3 (and Max Occurs = 10) Length Units = bytes	Left justify
Char[10][3][6]	Fixed Length	Length =6 (and Max Occurs = 30) Length Units = bytes	Left justify
Unsigned Char	Integer	Length = 1	Signed = no
Unsigned Char[2]	Integer	Length = 1 (and Max Occurs = 2)	Signed = no
Signed Char	Integer	Length = 1	Signed = yes
Signed Char[2]	Integer	Length = 1 (and Max Occurs = 2)	Signed = yes
Int	Integer	Length = 4	Signed = yes
Int[2]	Integer	Length = 4 (and Max Occurs = 2)	Signed = yes
Int[2][3]	Integer	Length = 4 (and Max Occurs = 6)	Signed = yes
Unsigned Int	Integer	Length = 4	Signed = no
Short	Integer	Length = 2	Signed = yes
Unsigned Short	Integer	Length = 2	Signed = no
Long	Integer	Length = 4 (see note 4)	Signed = yes
Long Long Int	Integer	Length = 8	Signed = yes
Float	Float	Length = 4	
Double	Float	Length = 8	
Long Double (see note 1)	Float	Length = 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 option does not import successfully; use the default 64 bit.
2. The length is affected by the *Address Size* C importer option:
   - For 32 bit, CWF length = 4 bytes.
   - For 64 bit, CWF length = 8 bytes.
3. The type and length of an enum is affected by the *Size of enum* C importer option:
   - For *1*: Logical type = xsd:byte, CWF physical type = Integer, CWF length = 1 byte.
   - For *2*: Logical type = xsd:short, CWF physical type = Integer, CWF length = 2 bytes.

- For *4*: Logical type = xsd:int, CWF physical type = Integer, CWF length = 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:
   - For 32 bit: CWF length = 4 bytes.
   - For 64 bit: CWF length = 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 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 COBOL importer uses a set of default values and behaviors when mapping COBOL data types to message model elements.

The following table shows how COBOL definitions influence the XML Schema settings in the message model.

COBOL Clause	XML Schema data type	Notes
PIC A	xsd:string	
PIC G	xsd:string	Set the compile-time locale name to ja_JP in **Window** → **Preferences** → **Importer** → **COBOL** to process this.
PIC N	xsd:string	Set the compile-time locale name to ja_JP in **Window** → **Preferences** → **Importer** → **COBOL** to process this.
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
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	

COBOL Clause	XML Schema data type	Notes
Any edited number	xsd:string	For example, a COBOL PICTURE clause that contains any of the following characters: 'Z' '+' '-' '.' ',' 'B' '0' or a currency symbol. If you want your broker logical type to be a numeric one, make sure that the COBOL PICTURE clause does not contain any of these characters.
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 MRM CWF characteristics of the elements that are generated in the message model.

COBOL Clause	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 G(n) PIC N(n)	Fixed Length String	Length = n Length Units = Characters	Justification = Left Justify Padding Character = SPACE
PIC 9(n) DISPLAY n=1-31	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 = 2, 4 or 8 based on n Length Units = Bytes	Signed = Unticked Sign Orientation = Blank
PIC 9(n) COMP-3 n=1-18	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 = Ticked Sign Orientation = Trailing *See Note 1

COBOL Clause	CWF Physical Type	CWF Length Characteristics	Other CWF characteristics
PIC S9(n) COMP or COMP-3 n=1-18	Integer or Packed Decimal	Length = See COMP and COMP-3 definitions above Length Units = Bytes	Signed = Ticked Sign Orientation = Blank
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 = 2, 4 or 8 for COMP Length Units = Bytes	Signed = Unticked Sign Orientation = Blank Virtual Decimal Point = n
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

Notes:

1. **Sign Orientation** can take one of the following values, based on the SEPARATE, LEADING, or TRAILING keywords in the COBOL definition:
   - Leading
   - Leading Separate
   - Trailing
   - Trailing Separate
2. The SYNC keyword causes the field to be aligned on a 1, 2, 4, or 8-byte boundary. This might cause 'slack bytes' to be added either before or after a field. **Leading Skip Count** is the number of such bytes that are added before a field; **Trailing Skip Count** is the number of such bytes that are added after a field.

   Leading Skip Count and **Trailing Skip Count** are calculated by the importer for each of the imported elements by the importer, irrespective of the SYNC clause. They have non-zero values when the SYNC clause is present.

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.

Refer to COBOL reference material for details of fields that require byte alignment.

3. All files that you import must be syntactically correct. Results are unpredictable if the file being imported is not synctactically correct.
4. COBOL data types that have keywords POINTER, COMP-X, INDEX, or PROCEDURE-POINTER, are not supported.
5. COBOL clauses that contain the keyword NATIVE cause an error, and are not imported.
6. COBOL level 66 and level 77 data items are not imported.
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 character string for PIC X, or to a decimal number for PIC 9.
8. If element names clash with Java language keywords, the element names are modified by prefixing the element name with a single underscore character.
9. Object-oriented extensions to COBOL 85 are not supported. For example, OBJECT-REFERENCE is not supported.
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.
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.

Signed external decimal numbers

The MRM Custom Wire Format (CWF) and TDS components of WebSphere Message Broker support the External Decimal (also known as Zoned Decimal) data format for numeric data. Numeric data in this format is stored internally as decimal character data. For example, in a system that uses the EBCDIC code, the number 1234 stored in a 4-byte external decimal field is stored as the character string '1234', and its actual internal hexadecimal representation is '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' can be used to indicate a positive value, and '0xB' or '0xD' can be 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') that is employed by IBM's pSeries machines, uses the normal ASCII codes ('0' [hex 30] to '9' [hex 39]) for the first or 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 method uses the same characters as an EBCDIC based machine, even though the actual internal hexadecimal representations of 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 copybook 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 that represents the external decimal field in the message (after the ASCII to EBCDIC, or EBCDIC to ASCII, conversion) maps to the code point that represents the correct sign for the decimal. This method includes the limitation that curly brace characters are variant (they have different code points in different EBCDIC code pages). This mechanism works only for those EBCDIC code pages 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 works for code page 500 but not for code page 871, where the curly braces have code points X'8E' and X'9C.

In an ASCII environment (determined by the CCSID property at run time), the default for both input and output is the 'Sign ASCII' representation. You can specify the applicable representation in the CWF physical layer for local attributes and local elements of types decimal, float, and integer.

Note: This option is only appropriate for those elements or attributes that have an external decimal physical representation, and that have an embedded ('Leading' or 'Trailing') sign (determined by the **Sign Orientation** property).

The table below shows 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. (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)

The next table gives some examples for a range of simple numbers that are representative of what can 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

Several objects are generated when you import from WSDL but restrictions might apply.

Generated objects

Files copied by command line import

The mqsicreatemsgdefsfromwsdl command copies the WSDL files it needs into the workspace before running the import process. These files are the

top level WSDL files and any imports resolved from a relative location. The files are copied under the specified message set into a folder called importFiles.

Report file
The WSDL importer appends the result of the import operation to a report file, listing all 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 files are added to the message set. Currently, the SOAP 1.1 definitions are always imported because:

- SOAP 1.1 is more widely used than SOAP 1.2.
- No standard SOAP 1.2 binding exists for WSDL 1.1. Therefore, the WSDL importer cannot reliably determine that SOAP 1.2 is required.
- You cannot import both SOAP 1.1 and SOAP 1.2 definitions because they use the same message name (for example, Envelope).

If you want to parse SOAP 1.2 instance documents, manually remove the SOAP 1.1 definitions and import the SOAP 1.2 definitions by using the Message Definition File wizard, selecting **IBM supplied message**.

If your message set has TDS or CWF layers, you might find that you get a number of warnings against the imported SOAP definitions. Most of these can be ignored, but take account of the allowed values for Boolean attributes. In SOAP 1.1 the Boolean values are 1 or 0, while 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 might 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 file.

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 shown in the following table.

	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"

	An input message	An output message
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 either the input or output element. 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 is global in the WSDL schema.

If the soap:body was defined with use="encoded" in the WSDL definition, 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".

WSDLs generated using .NET

In some instances, WSDLs that are generated using .NET include element references to the schema itself. An example of this type follows:

```
<xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="s:schema"/>
  </xsd:sequence>
</xsd:complexType>
```

For WSDLs of this type to be successfully imported into the Message Broker Toolkit without validation errors, you must manually add a namespace import statement to the namespace of the schema. An example of the import statement follows:

```
<xsd:import namespace="http://www.w3.org/2001/XMLSchema"/>
```

Place the import statement first within the schema element, and ensure it appears before any complex type or element definitions. Re-validate the WSDL by right-clicking the updated WSDL and clicking Validate.

Restrictions

Restrictions related to importing WSDL definitions exist where the WSDL definitions are not WS-I compliant.

SOAP Arrays

A WSDL 1.1 definition can define a SOAP Array (applicable only 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:

- 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] needs to 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 has 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. Therefore, 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>
```

You must manually edit the broker model created by importing the WSDL 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

A number of features in XML Schema are not supported, or their support is restricted in some way.

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 Redefine construct is accepted, but causes an error message to be displayed in the task list because it is not fully supported.

  The following XML shows an example of the Redefine construct:

```
<?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>
```

Message sets without namespace support

- 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>
```

Message model wizards

Wizards help to simplify complex message modeling tasks.

As software grows more complex, wizards are increasingly used to step you through complex tasks or procedures, ensuring that you correctly specify all the parameters that are required, and that you perform the required tasks in the correct order.

This topic provides some additional reference material for those wizards where you might need help in specifying certain parameters.

Each wizard that is documented here has its own high-level topic and a topic for each panel that is displayed by the wizard. The panels are listed in the order that they appear, and the fields on each panel appear in the topic in the same order as they appear on the panel. These topics provide only information about these fields and panels. You can find further information about the wizards in topics that are referenced from the wizard's high-level topic.

The following wizards have additional information:

- "New message definition file wizards"
- "Generate WSDL wizard" on page 825
- "Export WSDL wizard" on page 833
- "Configure New Web Service Usage wizard" on page 834

New message definition file wizards

Use the New message definition file wizards to create message definition files.

Depending on the selection that you make, you are routed through the correct sequence of panels to create the message definition file from the source that you have requested. Some panels are displayed only if certain conditions are met. These panels are marked as optional.

The following links provide further information about the panels and fields that form the New message definition file wizards.

- New Message Definition File (from scratch)
- 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 from scratch

Create a new message definition file by using the New message definition file wizard.

Create a new message definition file from scratch

When you create a new message definition file from scratch, you must set the following fields:

Message sets

Select the target message set
This field lists the message set projects that are available in your workspace. Click the down-arrow and select the appropriate message set from the list displayed. Depending on how you started the New message definition file wizard, a message set might be preselected for you, but this does not prevent you from selecting a different message set if you prefer.

Message definition file name
Specify the name of the message definition file that you are creating.

Schema for Schema settings

Prefix Specifies the namespace prefix to use for the namespace shown in the **Namespace** property.

Namespace
Specifies the namespace to be used.

Target namespace settings

Use target namespace
Selecting this check box allows you to specify a target namespace for the message definition file. You can choose a target namespace only if namespaces have been enabled in the message set.

Prefix 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: Create a new message definition file from an XML Schema file

You can create a new message definition file from an XML Schema file.

Create a new message definition file from an XML Schema file

When you create a new message definition file from an XML Schema file, you must set the following fields:

Select an XML Schema file

Message set
Use this field to choose the message set project that will contain the message definition file that you create.

Message definition file name
Use this field for the name of the message definition file that you want to create.

Select file from workspace
Choose this option if the XML Schema file that you want to add to the message definition file that you are creating is in the current workspace, and select the file from the displayed content of the workspace.

Select file from outside workspace
Choose this option if the XML Schema file that you want to add to the message definition file that you are creating is not in the current workspace, and specify the location of the XML Schema file that you want to add.

Copy source file into the 'importFiles' directory of the message set project
Select this check box to copy the source file into the 'importFiles' directory of the message set project.

Create an appropriate physical format if one does not already exist
Select this check box to automatically create the message set physical format that is needed by the pre-canned schema that you have selected for import.

New message definition file wizard: IBM supplied message

You can create a new message definition file from an IBM supplied message.

IBM supplied message wizard

When you create a message definition file from an IBM supplied message, you must set the following fields:

Select an IBM supplied message

Message set
Use this field to choose the message set project that will contain the message definition file that you create. Message set projects are filtered to only show artifacts in the active working set.

Message definition file name
Use this field for the name of the message definition file that you want to create.

IBM supplied messages
Select from the displayed set of IBM supplied message definitions. This field is split into two panes; the pane on the left displays the IBM supplied message definitions that are available, and the pane on the right contains text that gives advice about the usage of the message definition that you have selected in the field's left pane.

Copy source file into the importFiles directory of the message set project
Select this check box to copy the source file into the importFiles directory of the message set project.

Create an appropriate physical format if one does not already exist
Select this check box to automatically create the message set physical format that is required by the precanned schema that you have selected for import.

IBM supplied messages that you can import:

You can import IBM supplied messages 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 has an XML physical format layer, but does not have namespace support, the imported IBM supplied message is modified to remove namespaces. Therefore, enable namespace support before you import an IBM supplied message.
- If the message set to which you are adding the new message definition file does not have an XML physical format layer, but has namespace support, only the logical information is displayed in the model. The IBM supplied message is not modified to remove namespaces. Add an XML physical format to the message set before you import an IBM supplied message.
- If the message set to which you are adding the new message definition file does not have an XML physical 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 message definitions model the SOAP-defined portions of SOAP XML messages. They are best used with the SOAP parser. The definitions *Soap 1.1 Envelope* and *Soap 1.2 Envelope* model the SOAP envelope structure that is used to wrapper the user-defined body of a SOAP message. The definitions *Soap 1.1 Encoding* and *Soap 1.2 Encoding* model certain structures for use in "rpc/encoded" style SOAP messages.

  An IBM message for the SOAP domain tree is supplied as a schema that provides content-assist in creating a logical model for the SOAP domain by using the ESQL or mapping editor.

Multipart MIME message definitions
: These message definitions model the MIME-defined portions of multipart MIME messages and must be used with the message broker's MIME parser. Use the *MIME multipart header* definition for typical multipart MIME messages such as SOAP with Attachments, or RosettaNet. Use the *MIME Nested Multipart header* definition for multipart MIME messages in which the individual parts can themselves be multipart MIME; for example, S/MIME.

SAP IDoc message definitions
: These message definitions model the SAP-defined portion of ALE and File IDocs that precede the user-defined content. The ALE IDoc model can be used with the MRM and IDOC parsers. The File IDoc model can be used with the MRM parser only.

Timeout Request message definition
: This message definition models the TimeoutRequest message that is used in conjunction with the message broker TimeoutControl and TimeoutNotification nodes. You can use it with any parser.

CSV message definition
: This message definition models a CSV (comma separated value) format message. It can be used with the MRM parser.

New message definition file wizard: Create a new message definition file from an XML DTD file

You can create a new message definition file from an XML DTD file.

Create a new message definition file from an XML DTD file

When you create a new message definition file from an XML DTD file, you must set the following fields:

Select an XML DTD file

Message set
Use this field to choose the message set project that will contain the message definition file that you create. Message set projects are filtered to only show artifacts in the active working set.

Message definition file name
Use this field for the name of the message definition file that you want to create.

Select file from workspace
Choose this option if the XML DTD file that you want to add to the message definition file that you are creating is in the current workspace, and select the file from the displayed content of the workspace. XML DTD files are filtered to only show artifacts in the active working set.

Select file from outside workspace
Choose this option if the XML DTD file that you want to add to the message definition file that you are creating is not in the current workspace, and specify the location of the XML DTD file that you want to add.

Copy source file into the 'importFiles' directory of the message set project
Select this check box to copy the source file into the 'importFiles' directory of the message set project.

Create an appropriate physical format if one does not already exist
Select this check box to automatically create the message set physical format that is needed by the pre-canned schema that you have selected for import.

New message definition file wizard: Create a new message definition file from a C header file

You can create a new message definition file from a C header file.

Create a new message definition file from a C header file

When you create a new message definition file from a C header file, you must set the following fields:

Select a C header file

Message set
Use this field to choose the message set project that will contain the message definition file that you create. Message set projects are filtered to only show artifacts in the active working set.

Message definition file name
Use this field for the name of the message definition file that you want to create.

Target namespace
Use this field for the name of the target namespace for the message definition file that you want to create.

Select file from workspace
Choose this option if the C header file that you want to add to the message definition file that you are creating is in the current workspace, and select the file from the displayed content of the workspace. C header files are filtered to only show artifacts in the active working set.

Select file from outside workspace
Choose this option if the C header file that you want to add to the message definition file that you are creating is not in the current workspace, and specify the location of the C header file that you want to add.

Copy source file into the 'importFiles' directory of the message set project
Select this check box to copy the source file into the 'importFiles' directory of the message set project.

Create an appropriate physical format if one does not already exist
Select this check box to automatically create the message set physical format that is needed by the pre-canned schema that you have selected for import.

C header file options

Include paths

Preserve case in variable names
Select this check box if you want to preserve the case of the characters that form the names of the variables.

Select the pre-processing option to apply
Choose an option from the list.

New message definition file wizard: Create a new message definition file from a COBOL file

You can create a new message definition file from a COBOL file.

Create a new message definition file from a COBOL file

When you create a new message definition file from a COBOL file, you must set the following fields:

Select a COBOL file

Message set
Use this field to choose the message set project that will contain the message definition file that you create. Message set projects are filtered to only show artifacts in the active working set.

Message definition file name
Use this field for the name of the message definition file that you want to create.

Target namespace
Use this field for the name of the target namespace for the message definition file that you want to create.

Select file from workspace
Choose this option if the COBOL file that you want to add to the message definition file that you are creating is in the current workspace, and select the file from the displayed content of the workspace. COBOL files are filtered to only show artifacts in the active working set.

Select file from outside workspace
Choose this option if the COBOL file that you want to add to the message definition file that you are creating is not in the current workspace, and specify the location of the COBOL file that you want to add.

Copy source file into the 'importFiles' directory of the message set project
Select this check box to copy the source file into the 'importFiles' directory of the message set project.

Create an appropriate physical format if one does not already exist
Select this check box to automatically create the message set physical format that is needed by the pre-canned schema that you have selected for import.

COBOL file options

Preserve case in variable names
Select this check box if you want to preserve the case of the characters that form the names of the variables.

New message definition file wizard: Create a new message definition file from a WSDL file

You can create a new message definition file from a WSDL file.

Create a new message definition file from a WSDL file

When you create a new message definition file from a WSDL file, you must set the following fields:

Select a WSDL file

Message set
Use this field to choose the message set project that will contain the message definition file that you create. Message set projects are filtered to only show artifacts in the active working set.

Message definition file name
Use this field for the name of the message definition file that you want to create.

Select file from workspace
Choose this option if the WSDL file that you want to add to the message definition file that you are creating is in the current workspace, and select the file from the displayed content of the workspace. WSDL files are filtered to only show artifacts in the active working set.

Select file from outside workspace
Choose this option if the WSDL file that you want to add to the message definition file that you are creating is not in the current workspace, and specify the location of the WSDL file that you want to add.

Copy source file into the 'importFiles' directory of the message set project
Select this check box to copy the source file into the 'importFiles' directory of the message set project.

Create an appropriate physical format if one does not already exist
Select this check box to automatically create the message set physical format that is needed by the pre-canned schema that you have selected for import.

Add SOAP and XMLNSC to supported message domains if they do not exist
Select this check box to add the SOAP and XMLNSC message domains to the list of supported message domains that the message definition supports.

Generate WSDL wizard

The Generate WSDL wizard creates a WSDL definition from a message set.

The following links provide further information in relation to the panels and fields that form the Generate WSDL wizard. Some panels only appear if certain conditions are met. These are marked as (optional).

Open the Generate WSDL wizard

To open the Generate WSDL wizard:

1. Switch to the Broker Application Development perspective.
2. In the Broker Development view, right-click the folder that contains the message set file from which you want to generate a Web service definition, and select **Generate → WSDL Definition**. This starts the Generate WSDL wizard.

Generate WSDL wizard

The following panels are shown by the Generate WSDL wizard:

- Select the action you wish to perform
- Select a message set folder and destination directory
- Specify WSDL details
- Add operations to the WSDL definition
- Configure binding details - SOAP/HTTP
- Configure binding details - SOAP/JMS (optional)
- Summary of tasks

Generate WSDL wizard: Select the action you wish to perform

Use this panel to select how you want to generate the WSDL definition.

Generate WSDL wizard

The following panels are shown by the Generate WSDL wizard:

- Select the action you wish to perform
- Select a message set folder and destination directory
- Specify WSDL details
- Add operations to the WSDL definition
- Configure binding details - SOAP/HTTP
- Configure binding details - SOAP/JMS (optional)
- Summary of tasks

Panel properties

A choice of three options is presented.

Generate a new WSDL definition from existing message definitions
Select this option to generate a new WSDL definition from existing message definitions. This is the default option.

Export an existing WSDL definition to another directory in the workspace or file system.
Select this option to load the Export WSDL wizard.

Generate a new WSDL definition from existing message definitions using message categories (deprecated)
Select this option to generate a new WSDL definition using existing message definitions and message categories. This option is available to provide compatibility with previous releases of WebSphere Message Broker.

Generate WSDL wizard: Select a message set folder and destination directory

Use Generate WSDL wizard to select both the source of the WSDL definition and where you want the generated WSDL definition to be placed.

Generate WSDL wizard

The following panels are shown by the Generate WSDL wizard:

- Select the action you wish to perform
- Select a message set folder and destination directory
- Specify WSDL details
- Add operations to the WSDL definition
- Configure binding details - SOAP/HTTP
- Configure binding details - SOAP/JMS (optional)
- Summary of tasks

Panel properties

Select the message set folder from which to generate the WSDL definition:
Specify the message set folder from which to generate the WSDL definition.

Choose one of the following options to select the destination for the generated WSDL definition:

Create in a workspace directory
Select from your workspace directory the message set folder that will contain the generated WSDL definition.

Export to an external directory
Specify the address of the directory, outside your workspace, that you want to contain the generated WSDL definition.

Options
Specify the structure of the generated XML schema

Choose one of the following options to specify the structure of the generated XML schema:

Generate XML schema definitions with current directory structure
Generates the schema definition using the current directory structure; this is the default.

Generate XML schema definitions with flat structure
Generates the schema definition as a single level directory structure.

Generate WSDL wizard: Specify WSDL details

Use this panel to describe some details of the WSDL definition that you want to generate.

Generate WSDL wizard

The following panels are shown by the Generate WSDL wizard:

- Select the action you wish to perform
- Select a message set folder and destination directory
- Specify WSDL details
- Add operations to the WSDL definition
- Configure binding details - SOAP/HTTP
- Configure binding details - SOAP/JMS (optional)
- Summary of tasks

Panel properties

File Format

Select from:

- **Generate as a single WSDL file**

  The WSDL definition is written to a single file. This format is widely understood by external applications and SOAP toolkits.
- **Generate as a single WSDL file with all XML schema inlined**

  The WSDL definition is written to a single file with the XML added.
- **Generate as three WSDL files (one each for port type, service, and binding)**

  The WSDL definition is split into multiple files. This format offers better reuse of the component files.

WSDL Version

Select the required version of WSDL.

SOAP Version

Select the required version of SOAP

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 that are 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 that are described by the generated WSDL.

Generate WSDL wizard: Add operations to the WSDL details

Use this panel to define the operations that you want to add to the WSDL definition.

Generate WSDL wizard

The following panels are shown by the Generate WSDL wizard:

- Select the action you wish to perform
- Select a message set folder and destination directory
- Specify WSDL details
- Add operations to the WSDL definition
- Configure binding details - SOAP/HTTP
- Configure binding details - SOAP/JMS (optional)
- Summary of tasks

Panel properties

The panel is divided into two panes.

The top pane is read-only and displays a table that describes the operations that you have defined. The table has four columns with the following headings:

Operation
The name that you have given to the operation.

Input message
The name of the input message. This might be blank if the operation is a *Notification* type operation.

Output message
The name of the output message. This might be blank if no output message is specified for this operation.

Operation Type
The type of operation. Examples of operation type are:

Request-response

One-way

Solicit-response

Notification

The bottom pane is where you describe a new operation. The following fields describe the operation:

Name The name that you have given to the operation.

Operation Type
The type of operation. Examples of operation type are:

Request-response

One-way

Solicit-response

Notification

Input The name of the input message. This is omitted for a *Notification* operation.

Output
The name of the output message. This is omitted for a *One-way* operation.

Fault The name of the fault message. This is omitted for a *Notification* operation. Otherwise, you can specify one or more fault messages.

Generate WSDL wizard: Configure binding details - SOAP/HTTP

Use this panel to specify your SOAP/HTTP binding details, or to go to the panel that allows you to specify your SOAP/JMS binding details instead.

Generate WSDL wizard

The following panels are shown by the Generate WSDL wizard:

- Select the action you wish to perform
- Select a message set folder and destination directory
- Specify WSDL details
- Add operations to the WSDL definition
- Configure binding details - SOAP/HTTP
- Configure binding details - SOAP/JMS (optional)
- Summary of tasks

Panel properties

A choice of two options is presented:

SOAP/HTTP
Select this option to generate a new WSDL definition using existing message definitions. This is the default option.

SOAP/JMS
Select this option to load the WSDL export wizard.

If you select **SOAP/HTTP**, the following properties are displayed:

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.

If the WSDL definition is to contain multiple operations and they use different SOAP actions, you must add the unique SOAP action values to the WSDL after it has been generated. If all operations use the same SOAP action, specify the value here.

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. It must be a valid URL and it must include the port number, if this is different from the default HTTP port. An example of a port address is:
`http://localhost:9080/wassoap/servlet/router`

If you select **SOAP/JMS**, the SOAP/JMS panel is displayed. See "Generate WSDL wizard: Configure binding details - SOAP/JMS (optional)."

Generate WSDL wizard: Configure binding details - SOAP/JMS (optional)

Use this panel to specify your SOAP/JMS binding details.

Generate WSDL wizard

The following panels are shown by the Generate WSDL wizard:

- Select the action you wish to perform
- Select a message set folder and destination directory
- Specify WSDL details
- Add operations to the WSDL definition
- Configure binding details - SOAP/HTTP
- Configure binding details - SOAP/JMS (optional)
- Summary of tasks

Panel properties

This panel is displayed only when you have selected **SOAP/JMS** from the previous panel.

SOAP/HTTP
Select this option to generate a WSDL definition by using existing message definitions. This option is shown as not selected.

SOAP/JMS
Select this option to load the WSDL export wizard. This option is shown as selected.

The following properties are displayed for you to specify the bindings for SOAP/JMS:

Destination Style
This field is predefined as 'queue' and cannot be edited.

Initial Context Factory
The name of a Java class that allows 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 you can enter your own:

- com.ibm.websphere.naming.WsnInitialContextFactory. This class 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 class has a repository type of LDAP (Lightweight Directory Access Protocol)
- com.sun.jndi.fscontext.RefFSContextFactory. This class has a file system repository type.

The named class must be available on the classpath for a SOAP client or server that uses this WSDL binding.

If one of these classes 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 determines the Provider Type and additional details that you must provide in the JNDI Provider Type Properties section of this wizard panel.

JNDI Connection Factory
The JNDI name that is used to bind to the JMS connection factory; this property must match your JMS configuration.

JNDI Destination Name
The JNDI name for the JMS destination factory; this property must match your JMS configuration.

Host Name
The host name or IP address of the machine that is 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.

JNDI Provider URL
The resulting URL, used by JNDI; this property is read-only. It consists of the host name and port number and, optionally, the target context. For example, `iiop://hostname[:port] /[?TargetContext=ctx]`. The characters [] define optional content; do not include these characters in the string.

Service Name
The Service Name is 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 typically becomes the Java interface name.

Port Name
The name of a specific WSDL port for this service; it is typically derived from the Service Name. One convention is to provide a Service Name of <xyz> Service and a Port Name of <xyz> Provider.

The Port Name is 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 might become a Java class name.

Generate WSDL Definition wizard: Summary of tasks

Generate WSDL Definition wizard: provides a summary of the actions that will occur on finalizing the wizard.

Generate WSDL wizard

The following panels are shown by the Generate WSDL wizard:

- Select the action you wish to perform
- Select a message set folder and destination directory
- Specify WSDL details
- Add operations to the WSDL definition
- Configure binding details - SOAP/HTTP
- Configure binding details - SOAP/JMS (optional)
- Summary of tasks

Summary information

This panel consists of two panes. The top pane displays a summary of the selections you have made and the bottom pane lists the message definition files generated.

The selected message set
The message set you selected on the Select a message set folder and destination directory panel.

The generated WSDL files will go into:
The destination directory you selected on the Select a message set folder and destination directory panel.

The version of WSDL to be generated
The version you selected on the Specify WSDL details panel.

The version of SOAP to be generated
The version you selected on the Specify WSDL details panel.

The selected style for WSDL generation:
The style you selected on the Specify WSDL details panel.

The WSDL namespace:
The namespace you selected on the Specify WSDL details panel.

If you selected `rpc` as the style there is an entry for **RPC namespace**.

The following bindings are selected:

SOAP over HTTP
See Configure binding details - SOAP/HTTP for further details.

SOAP over JMS
See Configure binding details - SOAP/HTTP for further details.

The following WSDL files will be generated:
The name of the generated file.

Export WSDL wizard

The Export WSDL wizard exports a WSDL definition from a message set.

The following links provide further information in relation to the panels and fields that form the Export WSDL wizard.

Export WSDL wizard

The following panels are shown by the Export WSDL wizard:

- Select the WSDL definition you wish to export
- Specify the export location

Export WSDL wizard: Select the WSDL definition you wish to export

Use this panel of the Export WSDL wizard to select the WSDL definition that you want to export from a message set.

The following links provide further information in relation to the panels and fields that form the Export WSDL wizard.

Export WSDL wizard

The following panels are shown by the Export WSDL wizard:

- Select the WSDL definition you wish to export
- Specify the export location

Panel properties

The top pane of the panel shows a map of your workspace. Select the WSDL definition that you want to export.

Export file format

Choose one of the following options:

- **Export to a single WSDL file**

  The WSDL definition is written to a single file. This format is widely understood by external applications and SOAP toolkits.
- **Export to a single WSDL file with all XML schema inlined**

  The WSDL definition is written to a single file. This format is widely understood by external applications and SOAP toolkits.
- **Export to three WSDL files (one each for port type, service, and binding)**

  The WSDL definition is split into multiple files. This format offers better reuse of the component files.
- **Export based on the existing file structure**

  The WSDL definition is written to a single file. This format is widely understood by external applications and SOAP toolkits.

WSDL definition name

Select a name for the exported WSDL.

Export WSDL wizard: Specify the export location

Use this panel of the Export WSDL wizard to specify the location for the WSDL definition that you want to export from a message set.

The following links provide further information in relation to the panels and fields that form the Export WSDL wizard.

Export WSDL wizard

The following panels are shown by the Export WSDL wizard:

- Select the WSDL definition you wish to export
- Specify the export location

Panel properties

Choose one of the following options:

Export to a workspace directory
The structure of the workspace is displayed. Click the folder that you want the WSDL definition to be exported to.

Export to an external directory
Specify the name of the external directory that you want the WSDL definition to be exported to.

Select the **Overwrite existing files without warning** check box if you do not want to be warned that a file with the name that you specified is being overwritten. By default, the check box is cleared; if a file exists with the same name as the name that you have selected, you are prompted to confirm whether you want this file to be overwritten by the file that you are exporting.

Configure New Web Service Usage wizard

This provides additional reference information in relation to the Configure New Web Service Usage wizard.

You can launch this wizard by dragging deployable WSDL onto the message flow canvas.

The following links provide further information in relation to the panels and fields that form the Configure New Web Service Usage wizard.

Configure New Web Service Usage wizard

List of panels:

- Configure web service usage
- File generation details

Configure New Web Service Usage wizard: Configure Web service usage details

Use this panel of the Configure New Web Service Usage wizard to configure a new Web service.

The following links provide further information in relation to the panels and fields that form the Configure New Web Service Usage wizard.

Configure New Web Service Usage wizard

List of panels:

- Configure web service usage

- File generation details

Panel properties

Web service usage

Select from:

- **Expose message flow as Web service**

  The message flow is exposed as a Web service to its clients.

- **Invoke Web service from message flow**

  The Web service is invoked from the message flow.

Web service parameters

Configure the WSDL-related fields:

- **Port type**

  `Port type` must be specified, and lists all the port types defined in the WSDL document.

  By default, the drop down is populated with all the port types from the WSDL, in the order in which they appear in the WSDL file. The initially selected port type is the first port type that has at least one `http` binding associated with it.

- **Binding**

  `Binding` must be specified and lists all SOAP bindings with HTTP transport, associated with the selected port type.

  Bindings related to the selected port type are populated in the order in which they appear in the WSDL file. The initially selected binding is the one that has at least one port and one operation associated with it; if there is no such binding, the first binding with at least one port is selected.

  If no binding has ports associated with it, the first binding in the list is selected.

- **Service port**

  Lists all WSDL ports that point to the selected binding.

- **Binding operations**

  Lists all operations defined by the selected port type. Note, that only those operations implemented by the selected binding are selected by default.

  For every selected operation, the subflow generation process produces an output terminal, in the generated subflow.

  If you select an operation, that is not implemented by the selected binding, you receive a warning message; however, you can continue with the selection.

Configure New Web Service Usage wizard: File generation details

Use the Configure New Web Service Usage wizard to specify file generation details.

Configure New Web Service Usage wizard

List of panels:

- Configure web service usage
- File generation details

Panel properties

Flow Generation Details

Only one file is generated, namely the subflow. The subflow name is constructed as follows:

	Format of the generated subflow name
Request operation	`OperationName_WSDLFileName_MainFlow.msgflow`
Extract operation	`WSDLFileName_MainFlow.msgflow`

This page of the wizard lists the name of the file to be generated together with its location.

Typically, this file represents the subflow that is about to be generated. The default subflow name is prefixed by the name of the selected WSDL file, however, you have the option to change the name.

If the file to be generated already exists in the workspace, a warning is issued and the **Finish** button is no longer enabled.

You either have to change the name of the file, or select the **Overwrite existing file** check box.

Node type to be used by the Web service flows

Select from:

- **SOAP nodes**

  Select this option to use the SOAP domain and the SOAP nodes. This is the default option.

  Using SOAP nodes is WSDL driven and allows you to take advantage of various WS_* standards; for example WS_Security and WS_Addressing.

  If the message set does not support the SOAP domain you receive a warning.

- **HTTP nodes**

  Select this option if you want to use HTTP nodes rather than SOAP nodes.

  You can select this option only if the message set supports the XMLNSC, MRM, or XMLNS domains.

  If you select **HTTP nodes**, you see a message explaining the advantages of the SOAP nodes together with a suggestion that you import WSDL files.

  If you use the `ImportFiles` folder as your source, you can only select HTTP node generation.

Details

This pane appears if any additional warnings about the subflow that is generated apply. Possible warnings are as follows:

- When Service Definition is not found in the WSDL file, the URL property is not set on the node.
- You have selected one or more operations that are not implemented by the selected binding.
- When message domain is MRM, but XML wire format not found, message format property is not set on the HTTPInput or Request node.

Part 3. Appendixes

Appendix. Notices for WebSphere Message Broker

Read the legal notices for WebSphere Message Broker.

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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 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 in the WebSphere Message Broker Information Center

Review the trademark information for WebSphere Message Broker.

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at `www.ibm.com/legal/copytrade.shtml`.

Adobe is a registered trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Itanium, and Pentium are trademarks of Intel Corporation in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

F

G

H

I

K

L

M

N

P

Q

S

T

U

V

W

X

Printed in USA