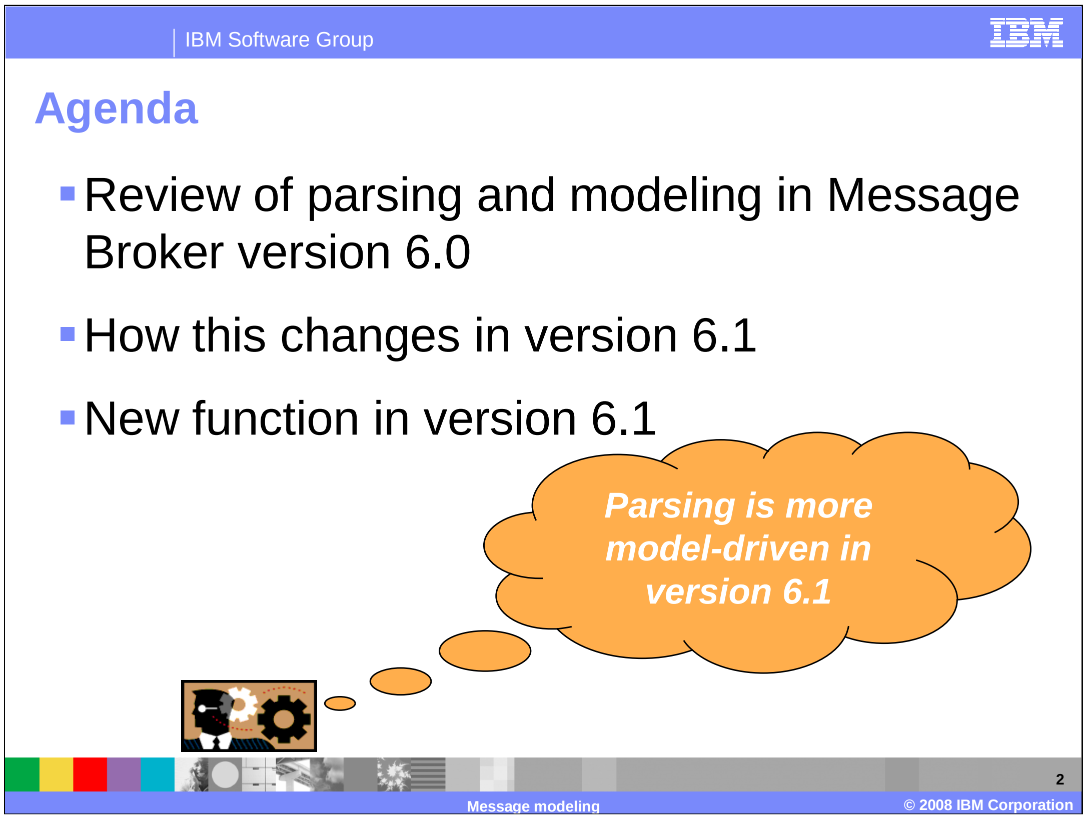
IBM Software Group

WebSphere® Message Broker Version 6.1

Message modeling

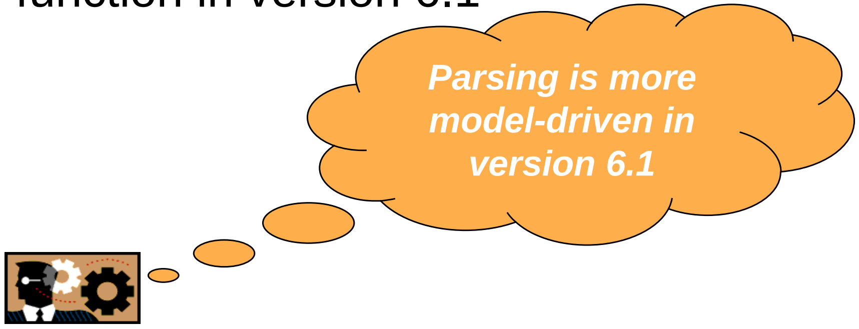
Updated January 22, 2008

This presentation provides a general introduction to message modeling in WebSphere Message Broker, and discusses the new facilities for modeling in Version 6.1.

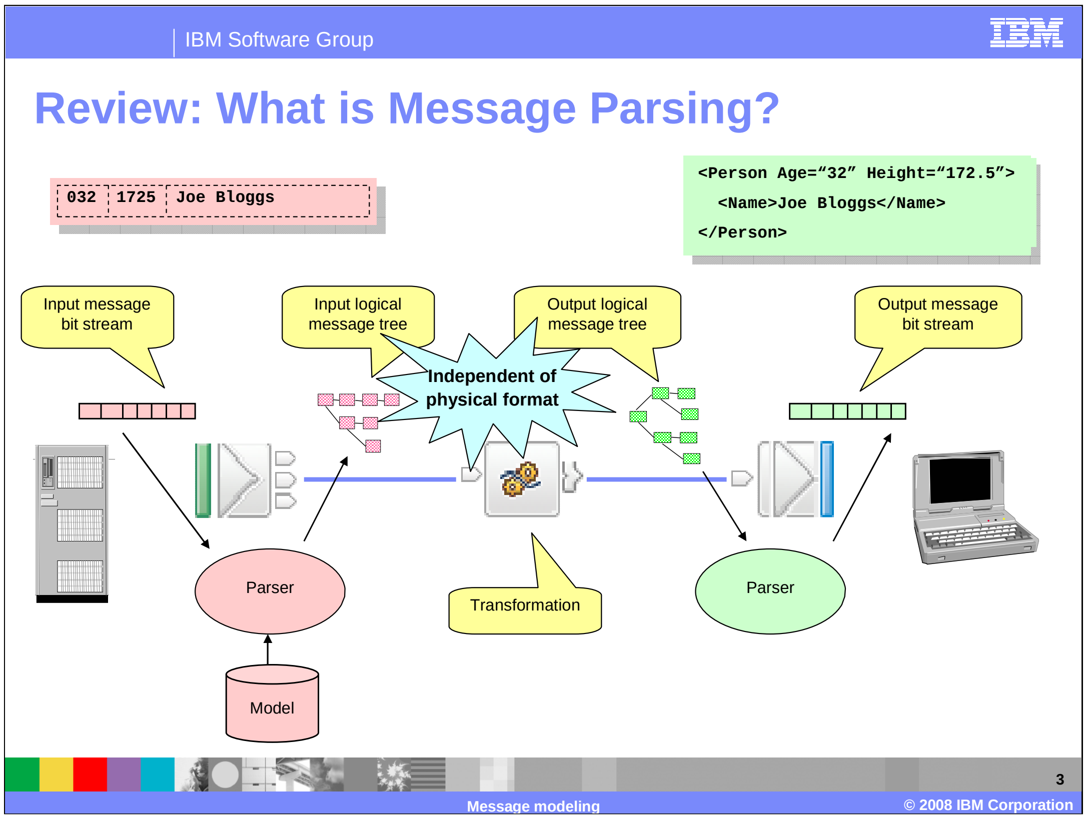


Agenda

- Review of parsing and modeling in Message Broker version 6.0
- How this changes in version 6.1
- New function in version 6.1



The presentation first provides a review of parsing and message modeling in Message Broker version 6.0. It then looks at how these basic techniques have evolved in version 6.1, and then concludes by looking at the new modeling function in version 6.1.



Review: What is Message Parsing?

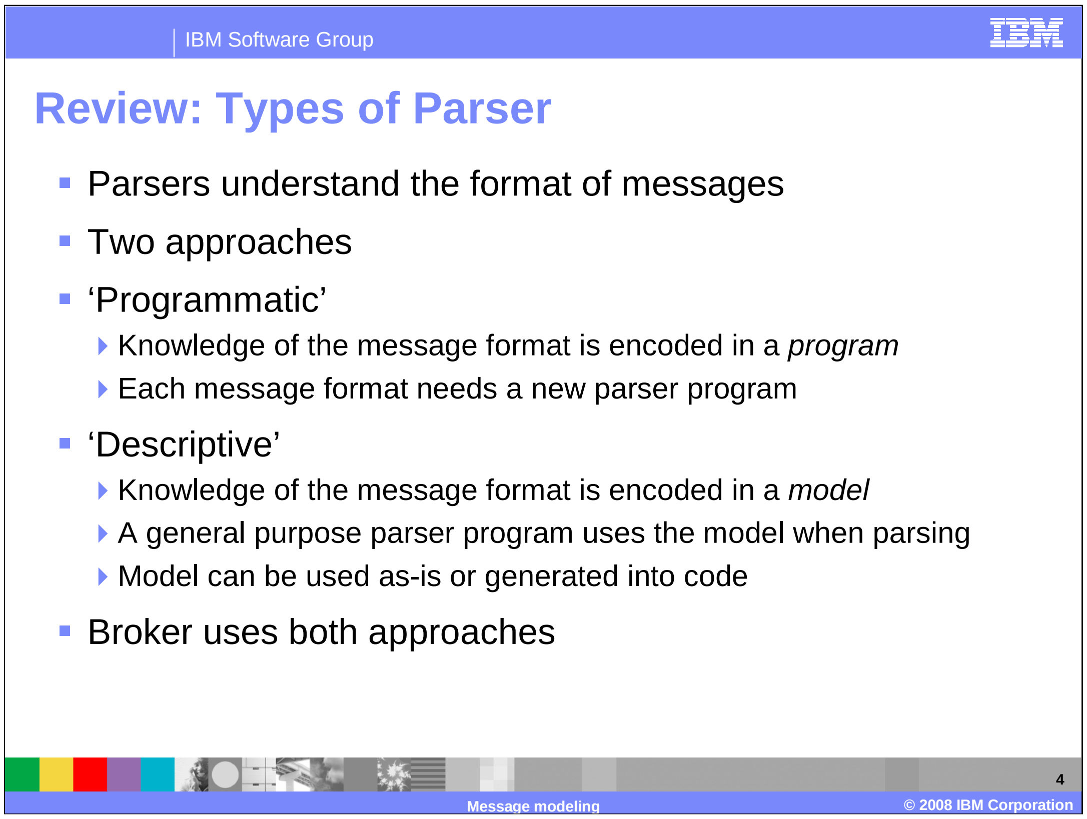
Typically a WebSphere Message Broker message flow will receive messages in a defined format, transform them, and output them in a different format. The example shows a COBOL Person data structure being transformed into an XML Person document. You can see that each of the Person messages contains Name, Age and Height fields.

The COBOL Person message arrives in the form of an *input bit stream*. Before it can be processed by the message flow, it must be converted into something called an *input logical message tree* for transformation by the message flow. This is a broker data structure that reflects the *logical structure* of the message. All of the broker's processing nodes work with the logical message tree, regardless of whether your transformation logic is expressed in Java or ESQL, or as a graphical mapping. In the example, some logic is applied which transforms the message and creates a new, output *logical message tree*. Finally this is converted into an *output bit stream* that represents the message as an XML document.

The component of the broker responsible for converting a bit stream into a logical message tree (*parsing*), and vice versa (*serializing* or *writing*), is called a *parser*. The parser must understand both the *physical format* of the bit stream and its logical structure in order to create the logical message tree. In the example you can see two parsers, one that understands COBOL data structures and another that understands XML.

Note that it is *only* the parser that needs to understand the physical format of the message. The logical message tree is *independent of the physical format* of the message bit stream. This decoupling of physical format from transformation logic is a key architectural feature of broker message flows.

A parser may, or may not, use a *model* when parsing. You will see the advantages provided by having such a model, later on.


IBM Software Group

Review: Types of Parser

- Parsers understand the format of messages
- Two approaches
- ‘Programmatic’
  - Knowledge of the message format is encoded in a *program*
  - Each message format needs a new parser program
- ‘Descriptive’
  - Knowledge of the message format is encoded in a *model*
  - A general purpose parser program uses the model when parsing
  - Model can be used as-is or generated into code
- Broker uses both approaches

4

Message modeling

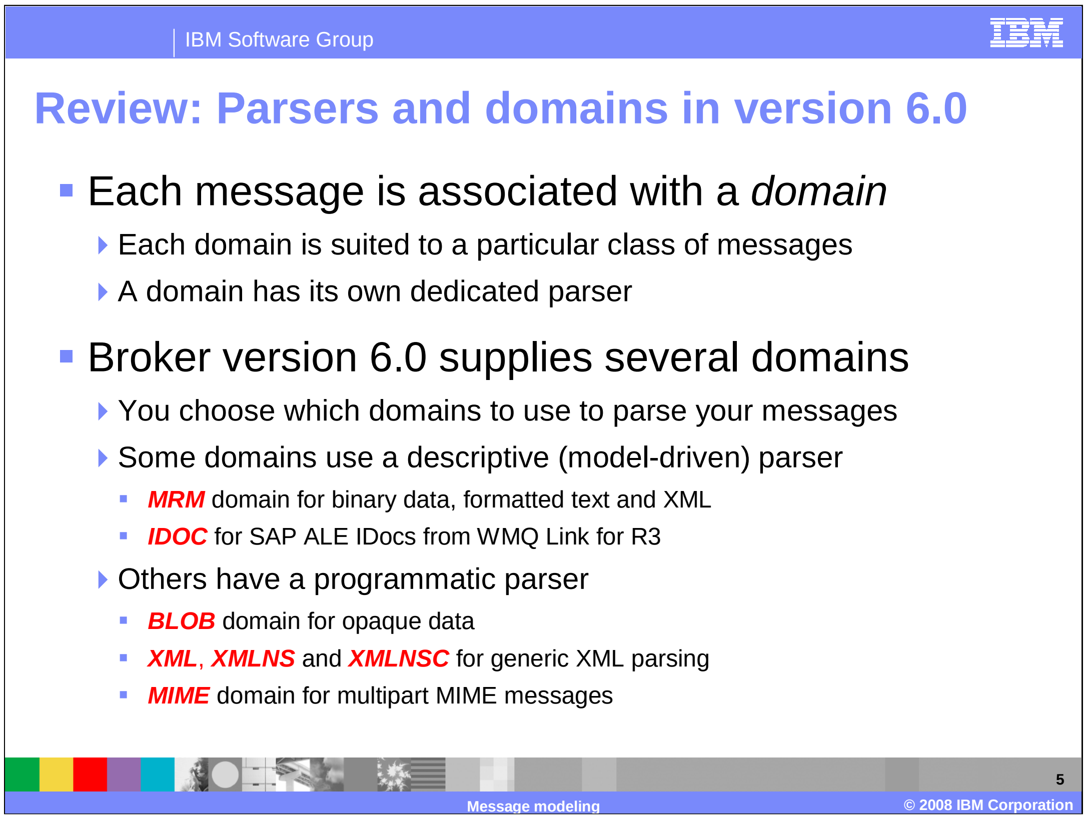
© 2008 IBM Corporation

A parser must understand the format of the messages that it will be parsing and serializing, it would not be of much use otherwise! There are two basic approaches to writing a parser.

The first is the ‘Programmatic’ approach, where a specific parser program would be written to parse each different message format. For example, if you had five different COBOL data structures, you would write five different format specific parsers. The knowledge of the format is hard-wired into the parser program code. This approach works well where the format tends to be fixed or where the format is self-defining. It has the advantage that optimizations for specific formats can easily be made, but has the disadvantage of inflexibility. A new parser program must be written for each new format, and if a format changes, the parser program must be changed, re-compiled, re-linked and re-deployed.

The second is the ‘Descriptive’ approach, where a single general purpose model-driven parser program would be written to parse all formats, each message format being represented by its own model. The knowledge of the format is not in the parser, instead it is in the model. The model is either accessed directly by the parser at runtime, or can be generated into code that the parser invokes. Model-driven parsing is a flexible approach as it is typically easier to change and deploy models, but it is harder to optimize a general purpose parser.

WebSphere Message Broker uses both programmatic and descriptive parsers, depending on the nature of the message formats involved.



Review: Parsers and domains in version 6.0

- Each message is associated with a *domain*
  - Each domain is suited to a particular class of messages
  - A domain has its own dedicated parser
- Broker version 6.0 supplies several domains
  - You choose which domains to use to parse your messages
  - Some domains use a descriptive (model-driven) parser
    - ***MRM*** domain for binary data, formatted text and XML
    - ***IDOC*** for SAP ALE IDocs from WMQ Link for R3
  - Others have a programmatic parser
    - ***BLOB*** domain for opaque data
    - ***XML***, ***XMLNS*** and ***XMLNSC*** for generic XML parsing
    - ***MIME*** domain for multipart MIME messages

When a message arrives at the broker it has to know how to parse it.

This is achieved by giving the message something called a domain.

Each domain is suited to a particular class of messages and has its own dedicated parser.

Each domain has a name, and its parser has the same name.

In fact the terms domain and parser are used interchangeably in the broker.

Message Broker supplies several domains and you choose the most suitable one for your message formats.

The way you tell the broker the domain is normally on the input node of the flow for an input message, or alternatively you can specify the domain in the MQRFH2 or JMS header.

For an output message you specify the domain when you create the output logical message tree.

Each message that is to be processed by a message flow must be associated with a *domain*. A domain determines the parser that is used when parsing and serializing the message. Each domain is suited to a particular class of messages. Some domains even support several different classes of message. The domain to use for an input message is typically specified on the input node of the message flow, but can also be specified by an MQRFH2 header or JMS header. The domain to use for an output message is specified when the message is created in the logical tree. To keep things simple, the parser for a domain has the same name as the domain – effectively domain and parser mean the same thing.

WebSphere Message Broker supplies several different domains. You choose the domain most suitable for the message format in question.

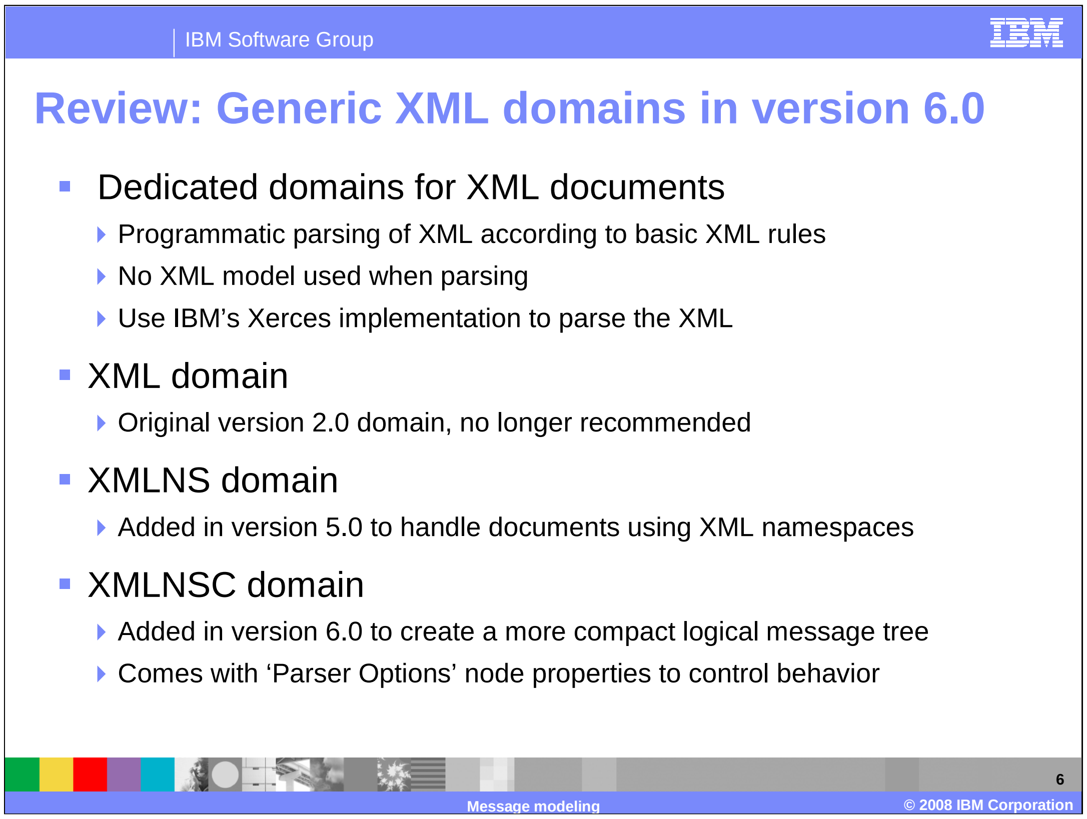
The most flexible domain is called "*MRM"* which has a general purpose model-driven parser capable of parsing binary messages, text messages and XML documents.

The *BLOB* domain can be used when the message is being treated as opaque, and just being routed. Effectively the message is not parsed at all.

For programmatic parsing of XML documents without a model, a set of 'generic' XML domains exist, called *XML*, *XMLNS*, and *XMLNSC.*

Specialist domains are provided for *MIME* messages (for example, SOAP with Attachments, RosettaNet) and for SAP *IDOC* structures.

If a supplied domain is not suitable for a particular message format, you can write your own programmatic or descriptive parser for use by your own user-defined domain.



Review: Generic XML domains in version 6.0

- Dedicated domains for XML documents
  - Programmatic parsing of XML according to basic XML rules
  - No XML model used when parsing
  - Use IBM's Xerces implementation to parse the XML
- XML domain
  - Original version 2.0 domain, no longer recommended
- XMLNS domain
  - Added in version 5.0 to handle documents using XML namespaces
- XMLNSC domain
  - Added in version 6.0 to create a more compact logical message tree
  - Comes with 'Parser Options' node properties to control behavior

A set of domains is provided for parsing XML documents without an XML model. Because XML is so verbose, a model is not actually needed to parse XML and a parser can be written to parse the XML programmatically.

The XML domain was the original way of processing XML documents in MQSI version 2.0. It has no support for XML namespaces and should not be used for developing new message flows.

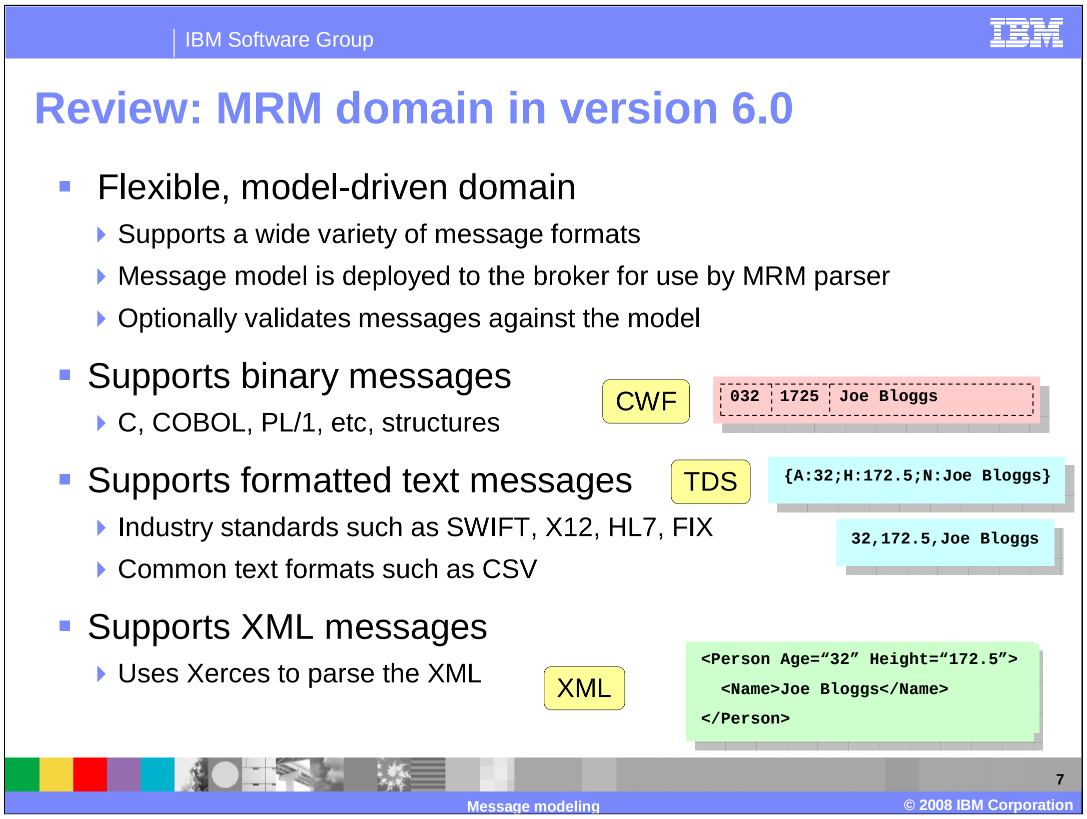
The XMLNS domain was added in Message Broker version 5 to provide true support for XML namespaces. Use of namespaces alters the content (but not the shape) of the logical message tree, which is why a new domain was created instead of altering the original XML domain.

Both XMLNS and XML create a logical tree whose shape conforms very closely to the XML data model. Hence, formatting white space is retained and simple elements and attributes have their value held in a separate child node of the tree.

The XMLNSC 'compact' domain was added in Message Broker version 6.0, to reduce the amount of memory occupied by logical message trees created from XML documents. This 'compact' behavior alters the shape of the logical message tree, which is why a new domain was created. By default, formatting white space is discarded and simple elements and attributes do not have child nodes. Substantial memory saving can result.

The XMLNSC domain is the first domain to exploit a facility in version 6.0 called parser options. These are a set of options exposed as node properties that are passed to a specific parser at runtime, thereby allowing its behavior to be controlled on a per-node basis.

Whether to use a generic XML domain or the MRM domain to parse your XML documents is discussed later on.



IBM Software Group

Review: MRM domain in version 6.0

- Flexible, model-driven domain
  - Supports a wide variety of message formats
  - Message model is deployed to the broker for use by MRM parser
  - Optionally validates messages against the model
- Supports binary messages
  - C, COBOL, PL/1, etc, structures
- Supports formatted text messages
  - Industry standards such as SWIFT, X12, HL7, FIX
  - Common text formats such as CSV
- Supports XML messages
  - Uses Xerces to parse the XML

CWF

```
032 | 1725 | Joe Bloggs
```

TDS

```
{A:32;H:172.5;N:Joe Bloggs}
```

```
32,172.5,Joe Bloggs
```

XML

```
<Person Age="32" Height="172.5">
  <Name>Joe Bloggs</Name>
</Person>
```

7

Message modeling

© 2008 IBM Corporation

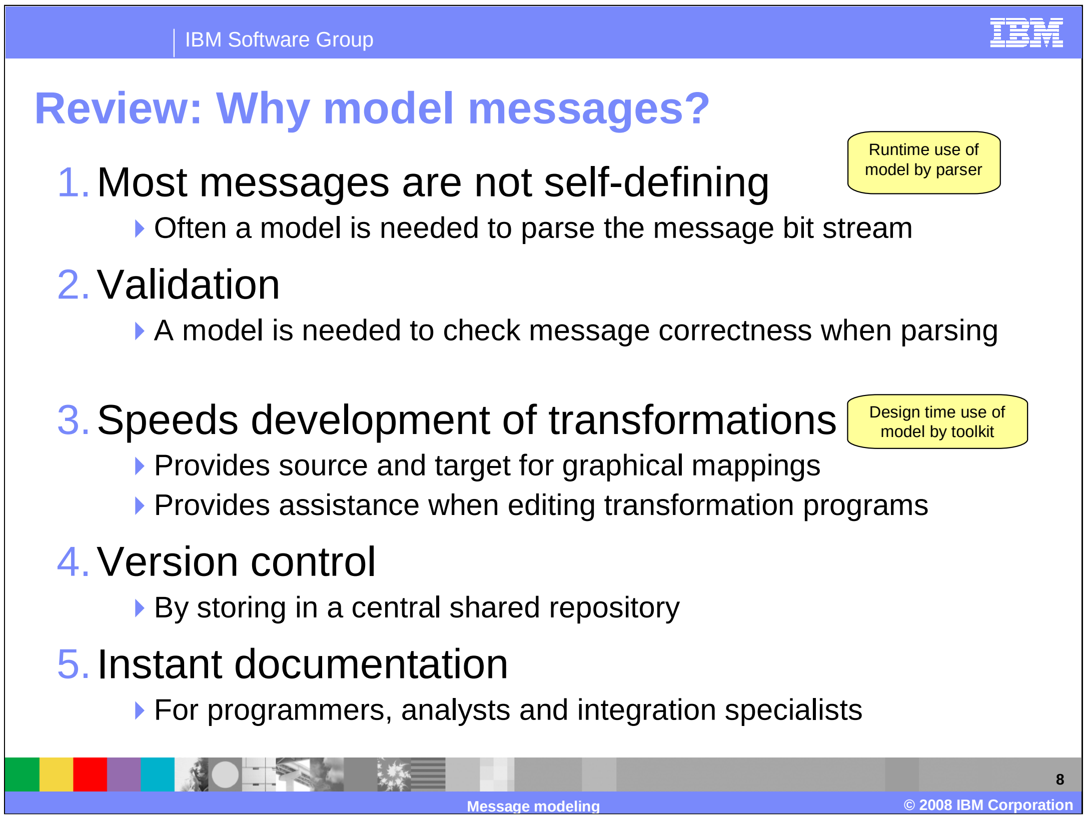
The most flexible domain is called MRM which has a general purpose model-driven parser. For example, the MRM parser can perform runtime validation of messages against the model.

The MRM domain supports modeling binary messages from applications written in C, COBOL, PL/1 and other languages. This support includes the ability to create a message model directly from a C header file or COBOL copybook. Binary messages are modeled using a physical format called *Custom Wire Format (CWF)*.

The MRM domain supports modeling formatted text messages, perhaps with field content identified by tags or separated by specific delimiters or both. This includes industry standards such as SWIFT, EDIFACT, X12, HL7 and FIX, and commonly used text messages such as Comma Separated Values (CSV). Text messages are modeled using a physical format called *Tagged/Delimited String* format *(TDS)*.

The MRM domain supports modeling XML messages, including those that exploit XML namespaces. Because it is using a model to parse XML, it provides extra functionality beyond the generic XML domains. Examples are creating objects in the logical message tree with the correct data type, and validating XML against the model. XML messages are modeled using a physical format called *XML Wire Format (XML)*.

Note that when parsing XML, the MRM behaves as a ‘compact’ domain like XMLNSC.


IBM Software Group

Review: Why model messages?

Runtime use of model by parser

1. Most messages are not self-defining
   - Often a model is needed to parse the message bit stream
2. Validation
   - A model is needed to check message correctness when parsing
3. Speeds development of transformations
   - Provides source and target for graphical mappings
   - Provides assistance when editing transformation programs
4. Version control
   - By storing in a central shared repository
5. Instant documentation
   - For programmers, analysts and integration specialists

Design time use of model by toolkit

8

Message modeling

© 2008 IBM Corporation

There are several reasons why you might need to model your messages. The first two are uses of the model by a *parser at runtime*. The others are uses of the model at *design time*, and are independent of whether the model is actually used by a parser.

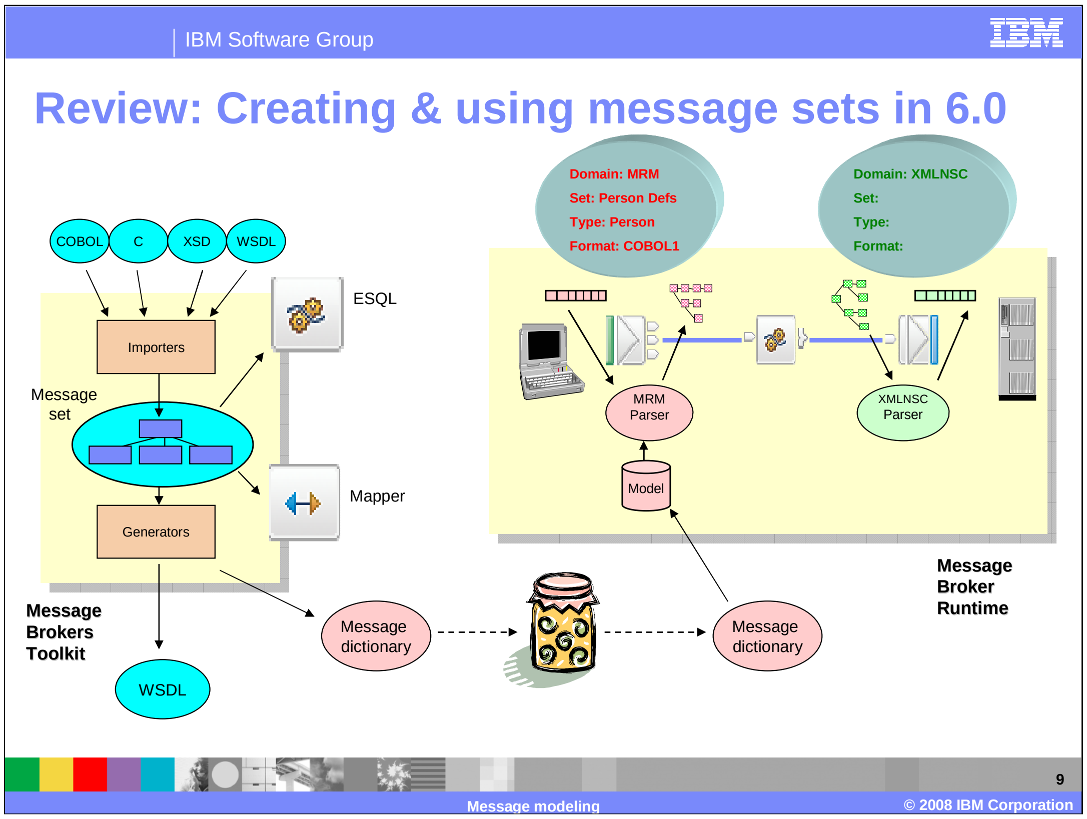
The majority of messages are not what is termed 'self-defining'. Think of a message created by a C or COBOL program – it's just a stream of binary data. Without some intelligence to interpret it, it is meaningless, and that's where the model comes in. XML is the opposite – it is so verbose that an XML parser can parse any XML document without using a model at all.

If you want to validate that your messages are correctly structured you need a model. An XML parser can parse any XML document but it can only check it is correct if it has a model to guide it.

A model can speed up the development of transformations enormously. For example, graphical mapping from a source message to a target message is not possible without a model. If you are transforming XML documents in this manner, you need a model even if the XML parser chooses not to use a model at runtime.

Models provide a good way of tracking different versions of your messages. COBOL programmers typically create a new version of a copybook each time a change is made, using a configuration control system. The same principle applies to message models.

A model provides documentation for a message that can be shared between programmers, business analysts and integration specialists.


IBM Software Group

Review: Creating & using message sets in 6.0

9

Message modeling

© 2008 IBM Corporation

Message models live in a container called a *message set*. The typical sequence of events when creating and using a message set is as follows.

First you create a message set project and message set.

Next you import application message formats described by XML DTD, XML Schema, WSDL types, C structures or COBOL structures, which create and populate *message definition files*. You can then edit the logical structure of your messages, and the physical format annotations, using the message definition editor.

Alternatively, you can create an empty message definition file and create your logical structure and physical annotations using just the editor.

If the message models in the message set are to be used by the MRM parser at runtime, you must deploy them to the broker. It is the message set that is the unit of deployment. You do this by adding the message set to a broker archive (.bar) file, which causes the message set to be generated into a compact form called a *message dictionary.*

If the message models in the message set are to be referenced by the Mapping editor or the ESQL editor, you must set the message flow project so that it references the message set project. This is done using the File, Properties menu.

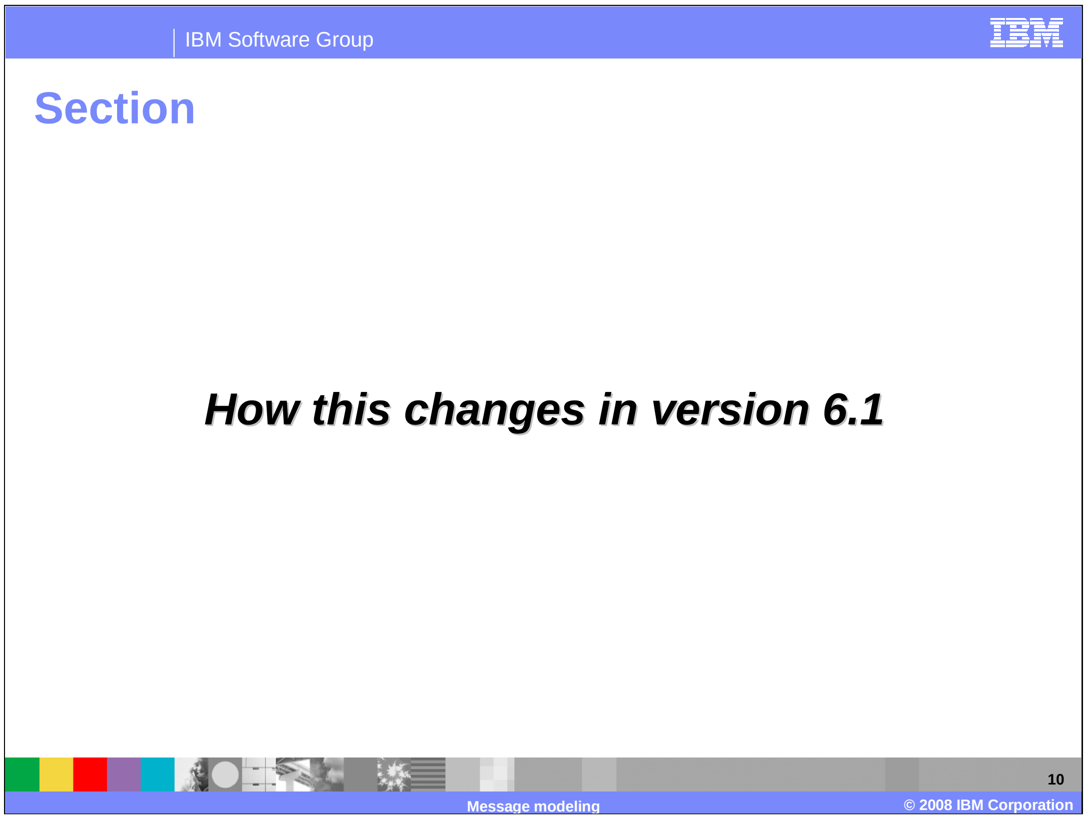
If the message models in the message set are to be used by other tools to create Web Services, you can generate WSDL from the message set using a wizard.

When using the MRM domain, in addition to associating the message with the domain as discussed earlier, you must also provide some further information. This is so the MRM parser can correctly locate the definition of the message in the model. These properties (*Set*, *Type*, *Format*) are specified in the same way as the domain.

The Set specifies the message set that contains the definition of the message.

The Type specifies the message within the message set. It should be set to the name of the message in the message model.

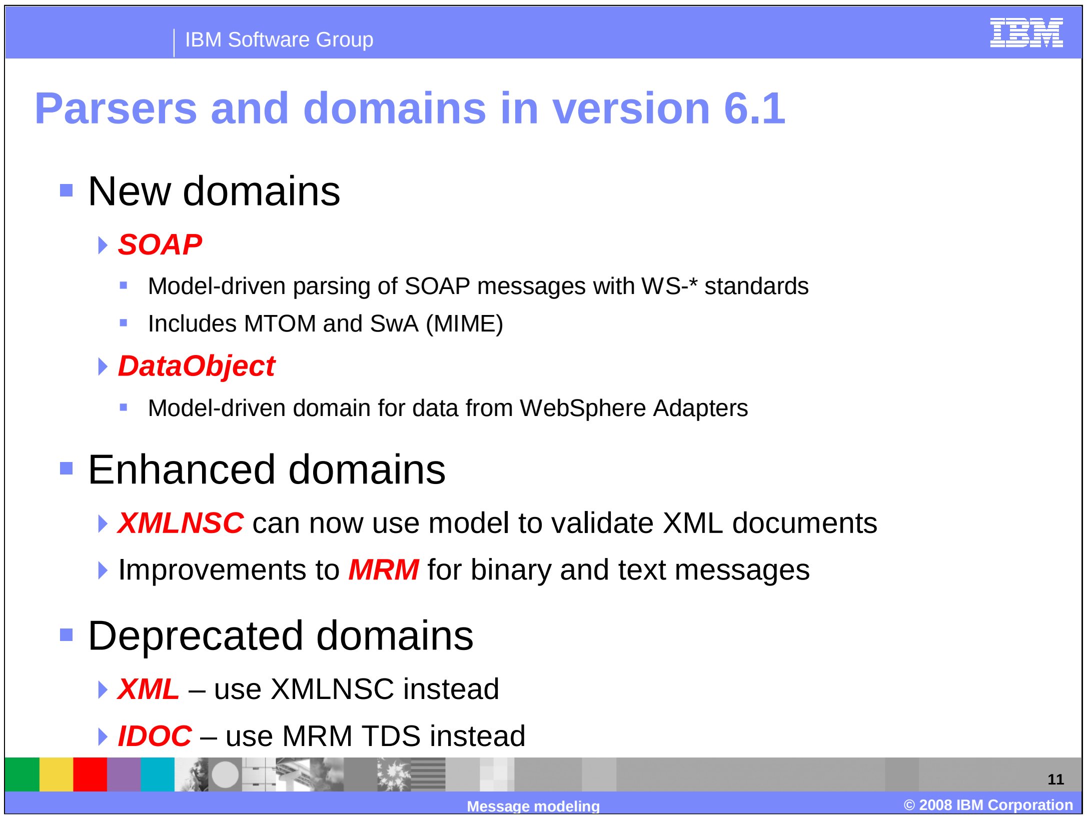
The Format specifies the physical format to use, which describes the physical layout of the message bit stream. It should be set to the name of the physical format in the message model.



Section

How this changes in version 6.1

The presentation now describes how this has changed in Message Broker version 6.1.



Parsers and domains in version 6.1

- New domains
  - *SOAP*
    - Model-driven parsing of SOAP messages with WS-* standards
    - Includes MTOM and SwA (MIME)
  - *DataObject*
    - Model-driven domain for data from WebSphere Adapters
- Enhanced domains
  - *XMLNSC* can now use model to validate XML documents
  - Improvements to *MRM* for binary and text messages
- Deprecated domains
  - *XML* – use XMLNSC instead
  - *IDOC* – use MRM TDS instead

Two new model-driven domains are added for version 6.1:

First, the SOAP domain. This is for use with the new SOAP nodes introduced in version 6.1. These nodes handle SOAP XML, MIME wrappered SOAP with Attachments and MTOM, and provide support for WS-Addressing and WS-Security standards. A canonical tree is created, which is owned by this new domain.

Second, the DataObject domain. This is for use with the new Adapter nodes. These nodes interface directly with enterprise information systems using embedded WebSphere adapters. A tree is created to represent the adapter business object, which is owned by this new domain.

Two key existing domains are enhanced:

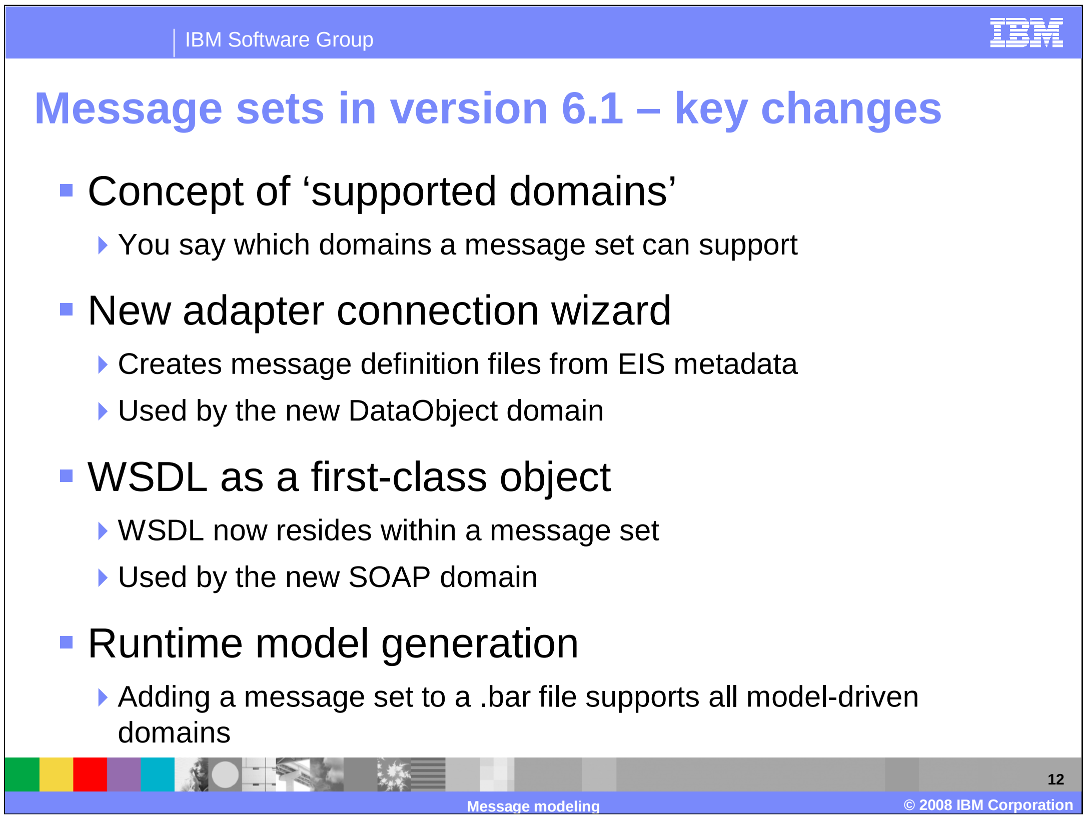
The ***XMLNSC*** domain is given a new internal high-speed parser which can be used without a model as in version 6.0, but can also parse and validate against a real XML Schema, so becoming model-driven. This is covered later in this presentation.

The ***MRM*** domain continues to be enhanced to increase the number of formats that can be modeled and to make modeling easier. Again, this is covered later in this presentation.

Two existing domains are officially deprecated:

XML domain. There is no need for this domain because XMLNSC or XMLNS can be used instead.

IDOC domain. There are limitations with this domain, and rather than fix them, MRM is now capable of handling SAP ALE and file IDocs, and so should be used instead.



Message sets in version 6.1 – key changes

- Concept of 'supported domains'
  - You say which domains a message set can support
- New adapter connection wizard
  - Creates message definition files from EIS metadata
  - Used by the new DataObject domain
- WSDL as a first-class object
  - WSDL now resides within a message set
  - Used by the new SOAP domain
- Runtime model generation
  - Adding a message set to a .bar file supports all model-driven domains

Message Broker version 6.1 made some changes to message set definitions. These are described in Part 1 of the Toolkit presentation.

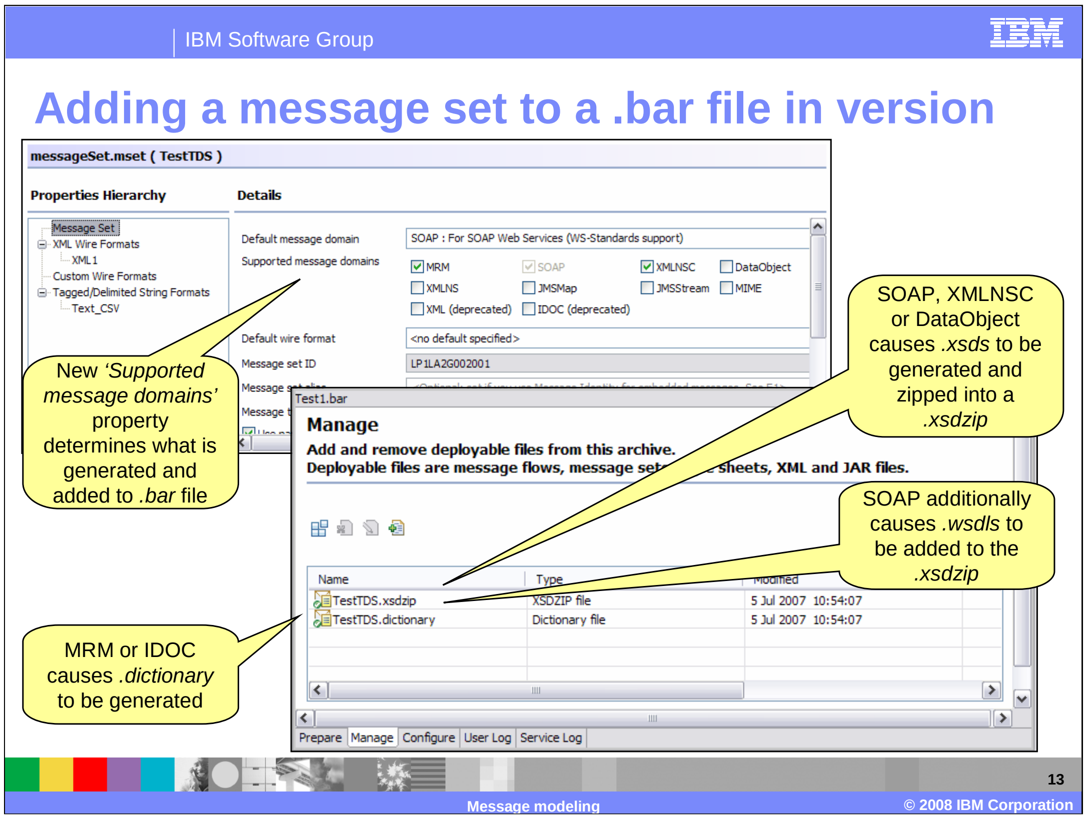
As a reminder, the key changes are:

First, you indicate in a message set which domains it is capable of supporting, using a new 'Supported message domains' property. There is no limit to the number of domains that can be supported, nor are there any combinations that are disallowed. However, best practice is to model unrelated models in separate message sets, so the recommendation is for most message sets to support one or two domains.

Second, a new Adapter Connection wizard allows you to interrogate EIS applications such as SAP and populate a message set with message definition files that model the messages that will be exchanged with the EIS applications. These models will be used by the new model-driven DataObject domain.

And finally, when WSDL is imported into a message set, the WSDL is added to the message set itself, becoming a first-class artifact within the message set. The WSDL will be used to configure new SOAP nodes and will be used by the new model-driven SOAP domain.

These changes are important to keep in mind when looking at the runtime models generated when a message set is added to a *.bar* file in version 6.1.



IBM Software Group

Adding a message set to a .bar file in version

messageSet.mset (TestTDS)

Properties Hierarchy

- Message Set
- XML Wire Formats
  - XML1
- Custom Wire Formats
- Tagged/Delimited String Formats
  - Text_CSV

Details

Default message domain	SOAP : For SOAP Web Services (WS-Standards support)
Supported message domains	[✓ MRM] [✓ SOAP] [✓ XMLNSC] DataObject XMLNS JMSMap JMSStream MIME XML (deprecated) IDOC (deprecated)
Default wire format	<no default specified>
Message set ID	LP1LA2G002001

Test1.bar

Manage

Add and remove deployable files from this archive.
Deployable files are message flows, message set sheets, XML and JAR files.

Name	Type	Modified
TestTDS.xsdzip	XSDZIP file	5 Jul 2007 10:54:07
TestTDS.dictionary	Dictionary file	5 Jul 2007 10:54:07

New *'Supported message domains'* property determines what is generated and added to *.bar* file

SOAP, XMLNSC or DataObject causes *.xsds* to be generated and zipped into a *.xsdzip*

SOAP additionally causes *.wsdls* to be added to the *.xsdzip*

MRM or IDOC causes *.dictionary* to be generated

13

Message modeling

© 2008 IBM Corporation

In version 6.1 the SOAP, DataObject, MRM, XMLNSC and IDOC domains are model-driven. Each domain requires its model to be deployed to the message broker runtime in an appropriate form.

Hence, the MRM and IDOC domains require message dictionaries, as in version 6.0.

The XMLNSC and DataObject domains each require an XML Schema.

The SOAP domain requires an XML Schema and WSDL.

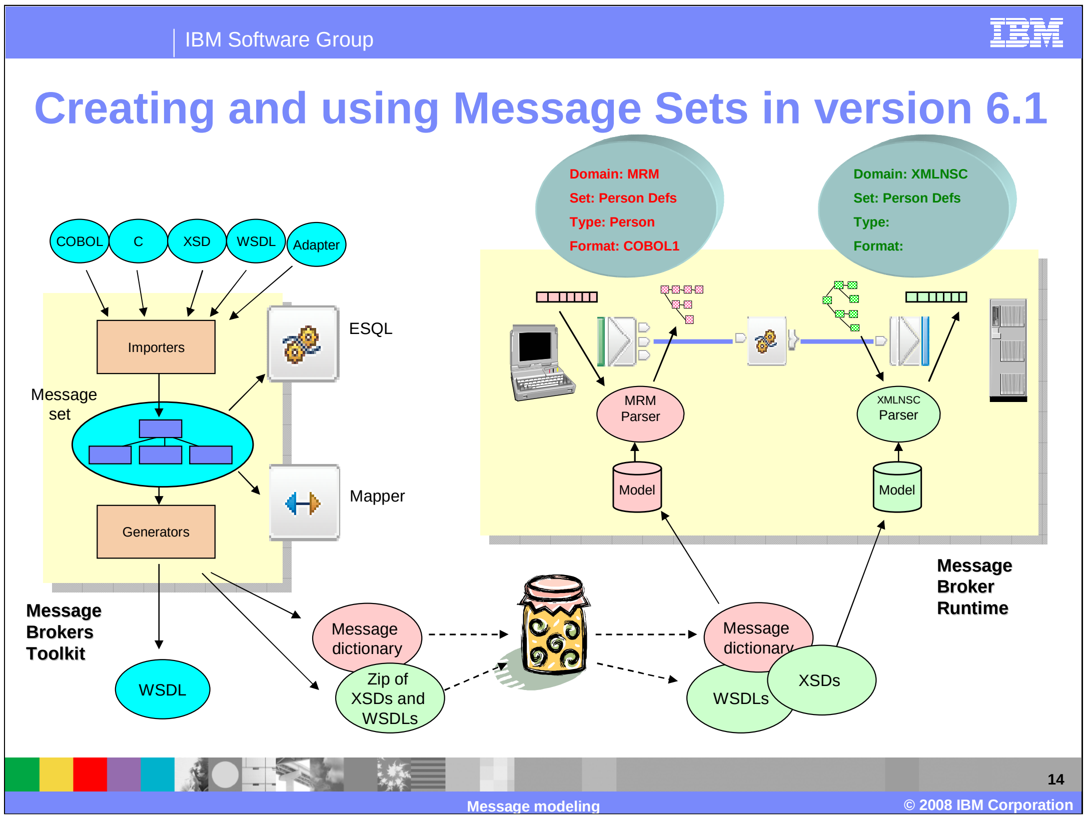
When a message set is added to a *.bar* file, it is the new *'Supported message domains'* property that determines which runtime models are generated and added to the *.bar.*

For MRM and IDOC domains a .dictionary file is generated from the message set as a whole and added to the *.bar*, as in version 6.0.

For XMLNSC, DataObject and SOAP domains, a .xsd file is generated from each message definition file. The *.xsd's* are archived into a single *.xsdzip* file, and the *.xsdzip* is added to the *.bar* file.

Additionally for SOAP, all *.wsdl* files are added to the *.xsdzip*.

Depending on the supported domains, a message set can therefore generate a *.dictionary*, a *.xsdzip*, or both.

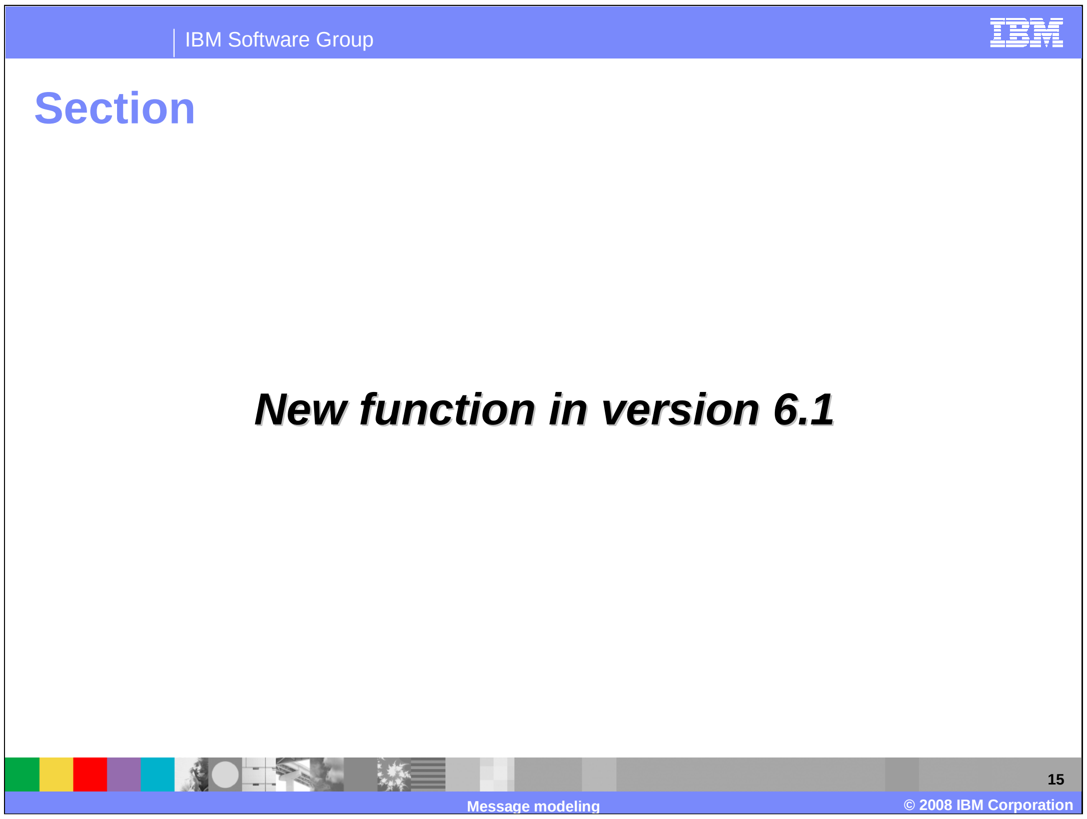

IBM Software Group

Creating and using Message Sets in version 6.1

How does this increased use of models affect creating and using message sets?

The most important difference is that you are now deploying real XML Schema and real WSDL files to the broker runtime. If the message set is flagged as supporting XMLNSC or SOAP or DataObject domains, then when the message set is added to the .bar file, a .xsdzip file is created. This holds the generated XML Schema and the original WSDL files. This is extracted on the broker and saved on the broker’s file system according to message set.

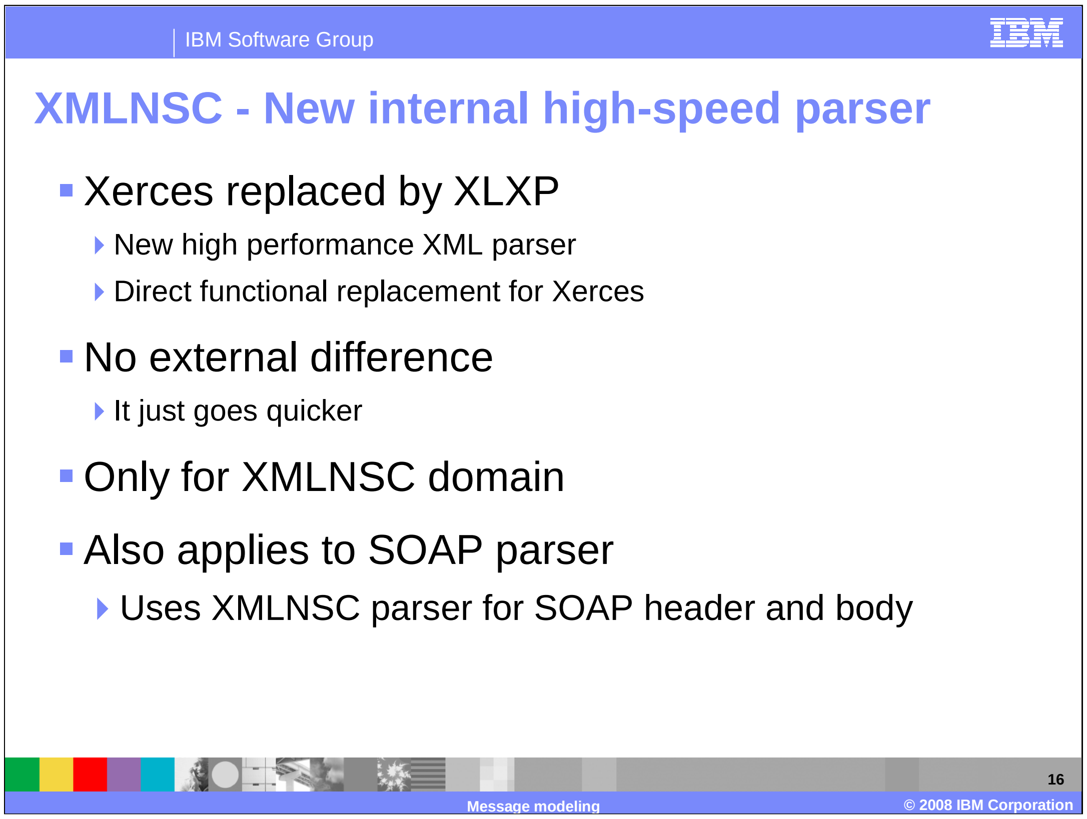
When you specify that a message flow is to use XMLNSC or SOAP or DataObject as the message domain, you must also supply the name of the message set. The parser for the domain loads the XML Schema or WSDL and uses it appropriately.



Section

New function in version 6.1

This section covers the new function introduced in Message Broker version 6.1 for Message Modeling.


IBM Software Group

XMLNSC - New internal high-speed parser

- Xerces replaced by XLXP
  - New high performance XML parser
  - Direct functional replacement for Xerces
- No external difference
  - It just goes quicker
- Only for XMLNSC domain
- Also applies to SOAP parser
  - Uses XMLNSC parser for SOAP header and body

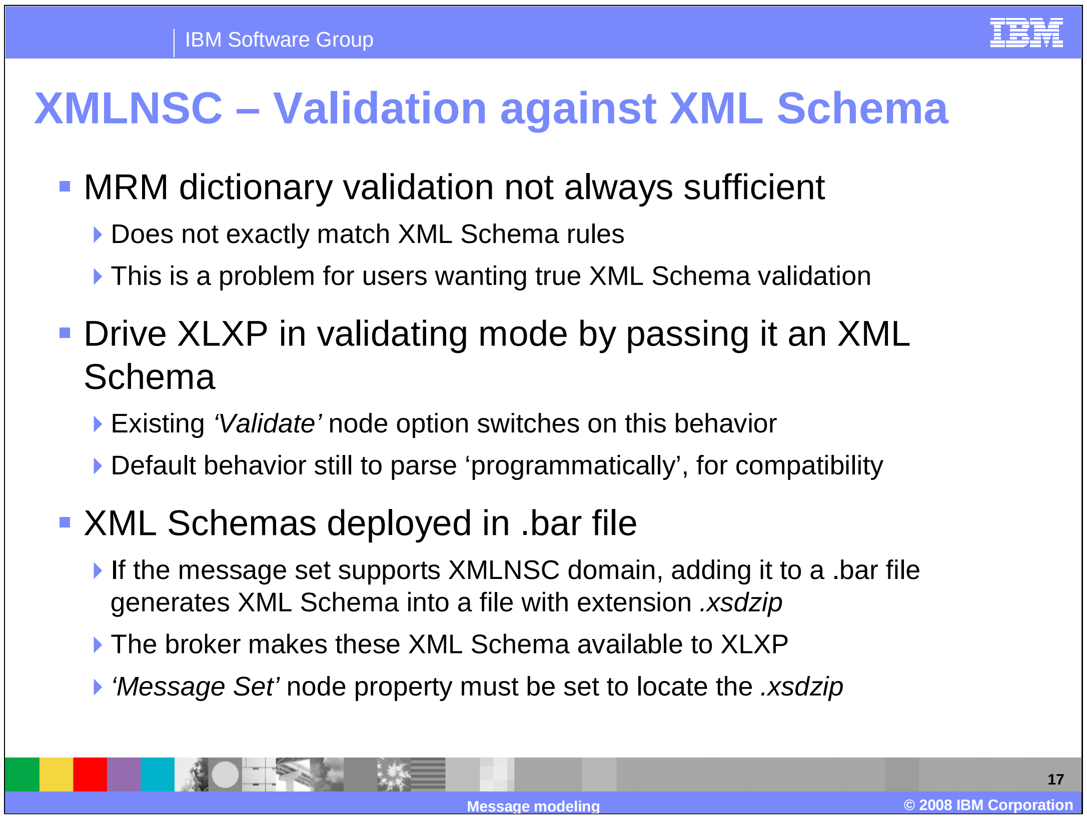
16

Message modeling

© 2008 IBM Corporation

There have been several important changes to the way that the XMLNSC domain operates.

First, a faster XML parser called XLXP has replaced IBM's Xerces parser. The XMLNSC domain has switched to use XLXP with a consequent improvement in performance. This switch has no external impact, and version 6.0 message flows that specify XMLNSC will operate unchanged when migrated to version 6.1. Note that this change has only been made to XMLNSC, not to XML, XMLNS or MRM XML, which continue to use Xerces. The new SOAP parser uses the XMLNSC parser internally.


IBM Software Group

XMLNSC – Validation against XML Schema

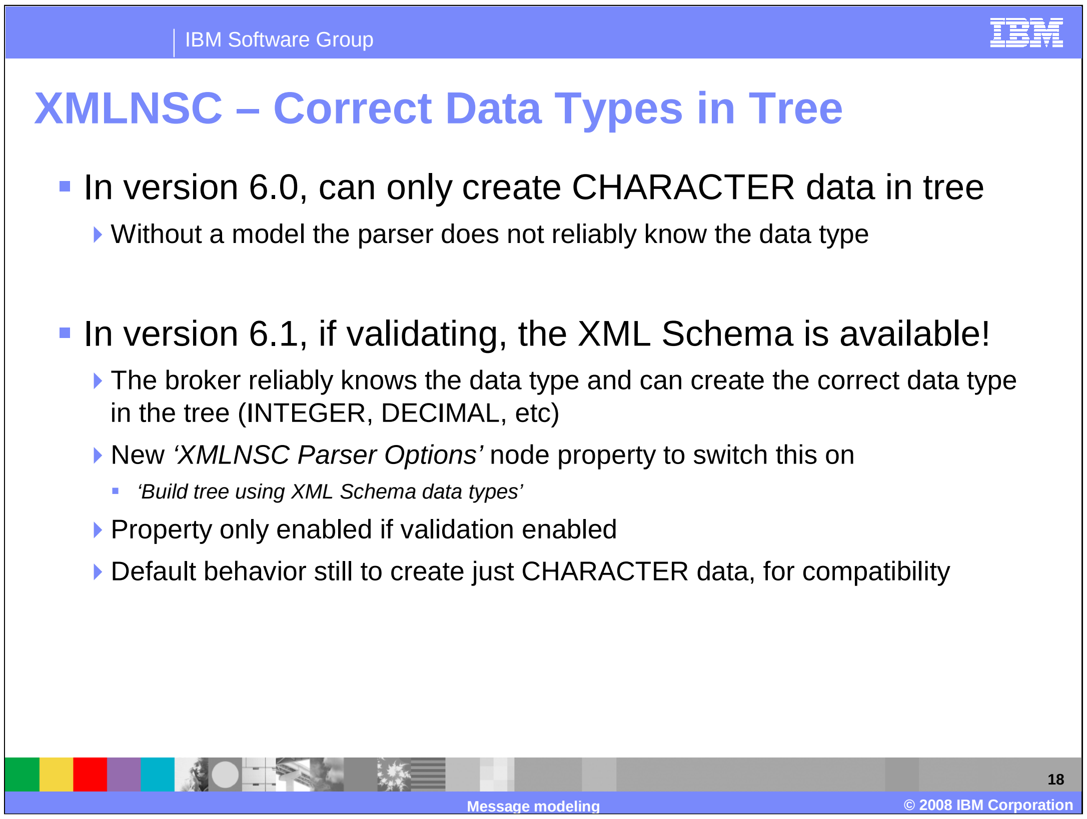
- MRM dictionary validation not always sufficient
  - Does not exactly match XML Schema rules
  - This is a problem for users wanting true XML Schema validation
- Drive XLXP in validating mode by passing it an XML Schema
  - Existing *'Validate'* node option switches on this behavior
  - Default behavior still to parse 'programmatically', for compatibility
- XML Schemas deployed in .bar file
  - If the message set supports XMLNSC domain, adding it to a .bar file generates XML Schema into a file with extension *.xsdzip*
  - The broker makes these XML Schema available to XLXP
  - *'Message Set'* node property must be set to locate the *.xsdzip*

17

Message modeling © 2008 IBM Corporation

Second, validation against real XML Schema. Because the MRM validates against its own dictionary model, it does not 100% faithfully reproduce XML Schema validation semantics. This requirement is addressed in version 6.1, as the XMLNSC domain can now validate against real XML Schema. Whether to validate is controlled by the existing node '*Validate'* option, just like the MRM. If validation is selected, then you must also supply the name of the message set which is used to locate the XML Schema on the broker's file system. XMLNSC validation considerably out-performs MRM validation too, as the architecture of the XLXP parser is optimized for the model-driven scenario.

It's important to note that XMLNSC therefore operates in one of two modes. The default for version 6.0 compatibility is to operate in non-validating mode. If validation is selected then it operates in validating mode and expects a *Message Set* property to be supplied. The *Message Type* property is not needed as the name of the root element in the document is used. *Message Format property* is not needed as only the logical model is used by XMLNSC.

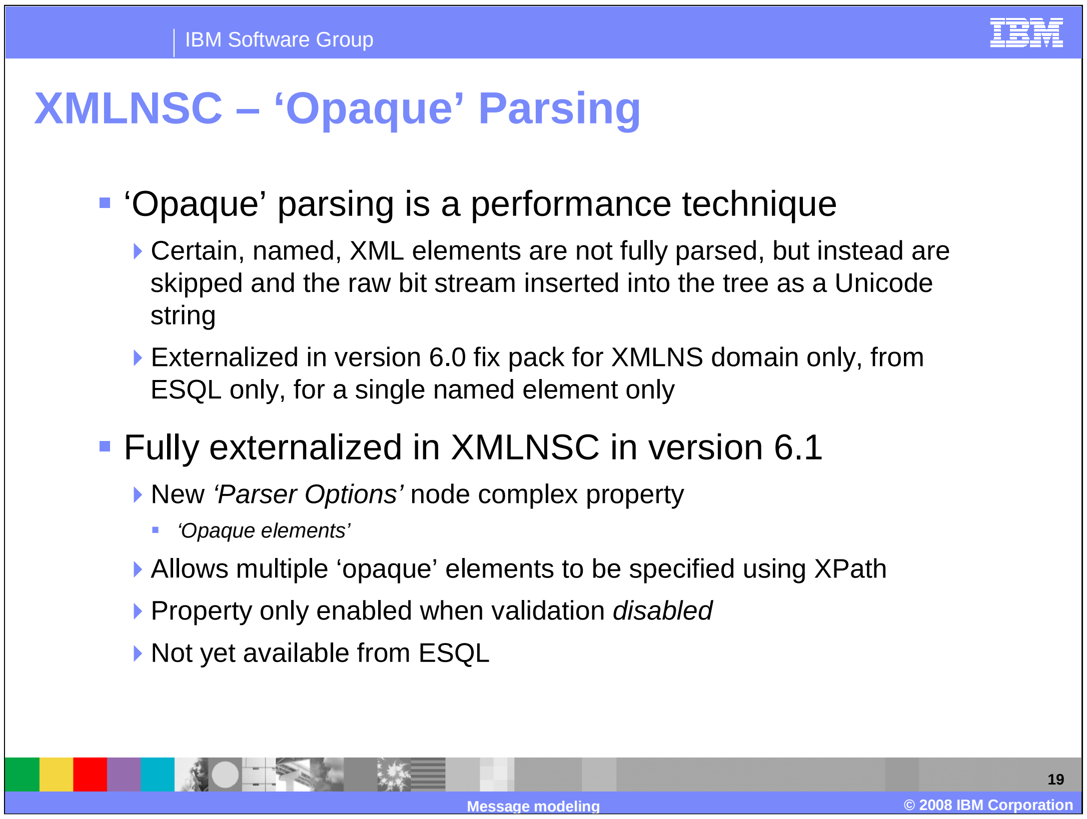


XMLNSC – Correct Data Types in Tree

- In version 6.0, can only create CHARACTER data in tree
  - Without a model the parser does not reliably know the data type
- In version 6.1, if validating, the XML Schema is available!
  - The broker reliably knows the data type and can create the correct data type in the tree (INTEGER, DECIMAL, etc)
  - New *‘XMLNSC Parser Options’* node property to switch this on
    - *‘Build tree using XML Schema data types’*
  - Property only enabled if validation enabled
  - Default behavior still to create just CHARACTER data, for compatibility

In version 6.0, XMLNSC never used a model and so had to assume that all XML data it encountered was of character data type. Accordingly all syntax elements it created in the tree were of ESQL data type CHARACTER. In version 6.1, if XMLNSC is operating in validating mode, it has the XML Schema and therefore knows the XML Schema data type of the XML data. This enables syntax elements to be built that have a data type that matches the XML Schema data type, in the same way as the MRM builds its tree.

The default for version 6.0 compatibility is to build a tree with CHARACTER data types only. If a new option called *‘Build tree using XML Schema data types’* is selected then the tree will be built with appropriate data types. This new option can be found under *‘XMLNSC Parser Options’* and is only enabled in *validating* mode.

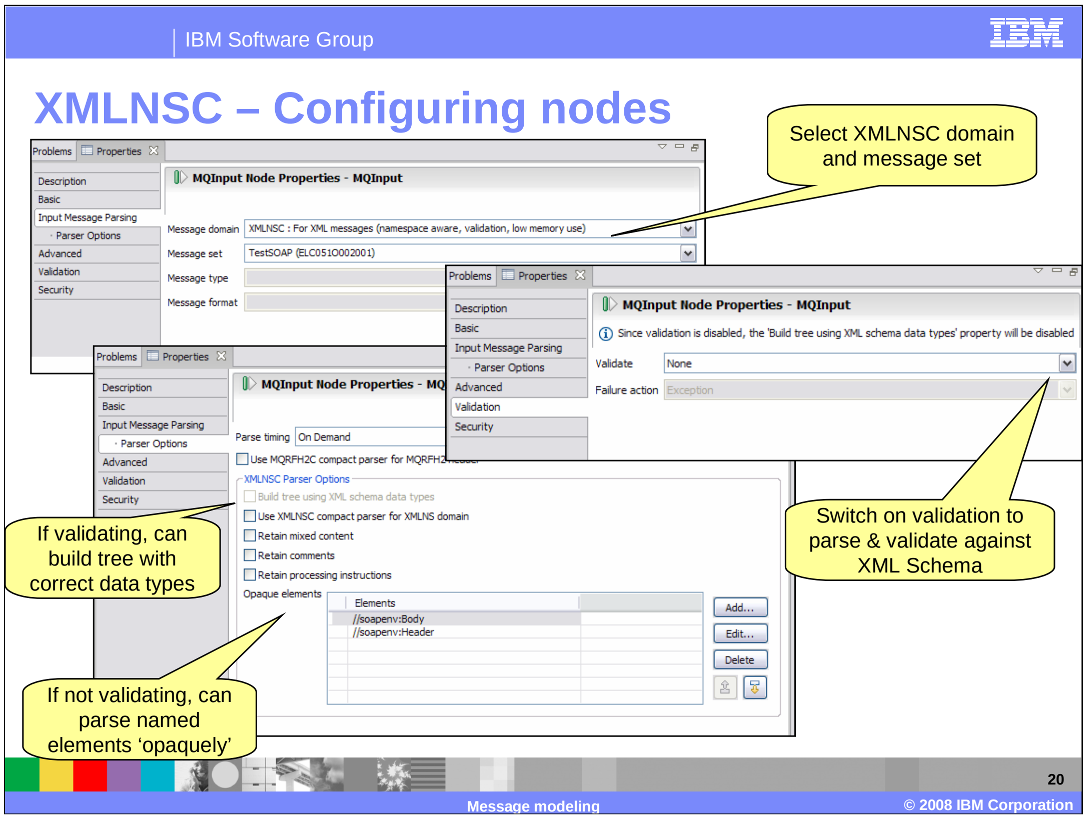


XMLNSC – 'Opaque' Parsing

- 'Opaque' parsing is a performance technique
  - Certain, named, XML elements are not fully parsed, but instead are skipped and the raw bit stream inserted into the tree as a Unicode string
  - Externalized in version 6.0 fix pack for XMLNS domain only, from ESQL only, for a single named element only
- Fully externalized in XMLNSC in version 6.1
  - New *'Parser Options'* node complex property
    - *'Opaque elements'*
  - Allows multiple 'opaque' elements to be specified using XPath
  - Property only enabled when validation *disabled*
  - Not yet available from ESQL

Opaque parsing is a term used to describe a performance enhancement, where named XML elements are not fully parsed, but instead are skipped and the raw bit-stream inserted into the tree as a Unicode string. It's available in version 6.0, but only exposed in a limited way. In version 6.1 it has been fully externalized through a new option called *'Opaque elements', which* allows you to specify the XPaths of the elements you want to parse opaquely. This new option can be found under *'XMLNSC Parser Options',* and is only enabled in *non-validating* mode, because by definition validation involves fully expanding all elements.

The SOAP domain uses XMLNSC domain internally to parse SOAP XML, so all these capabilities apply to SOAP domain too.



IBM Software Group

XMLNSC – Configuring nodes

Select XMLNSC domain and message set

Problems | Properties

Description
Basic
Input Message Parsing
· Parser Options
Advanced
Validation
Security

MQInput Node Properties - MQInput

Message domain: XMLNSC : For XML messages (namespace aware, validation, low memory use)
Message set: TestSOAP (ELC051O002001)
Message type:
Message format:

Problems | Properties

MQInput Node Properties - MQInput

Since validation is disabled, the 'Build tree using XML schema data types' property will be disabled

Validate: None
Failure action: Exception

Switch on validation to parse & validate against XML Schema

Problems | Properties

MQInput Node Properties - MQ

Parse timing: On Demand
☐ Use MQRFH2C compact parser for MQRFH2 header

XMLNSC Parser Options
☐ Build tree using XML schema data types
☐ Use XMLNSC compact parser for XMLNS domain
☐ Retain mixed content
☐ Retain comments
☐ Retain processing instructions

Opaque elements

Elements
//soapenv:Body
//soapenv:Header

If validating, can build tree with correct data types

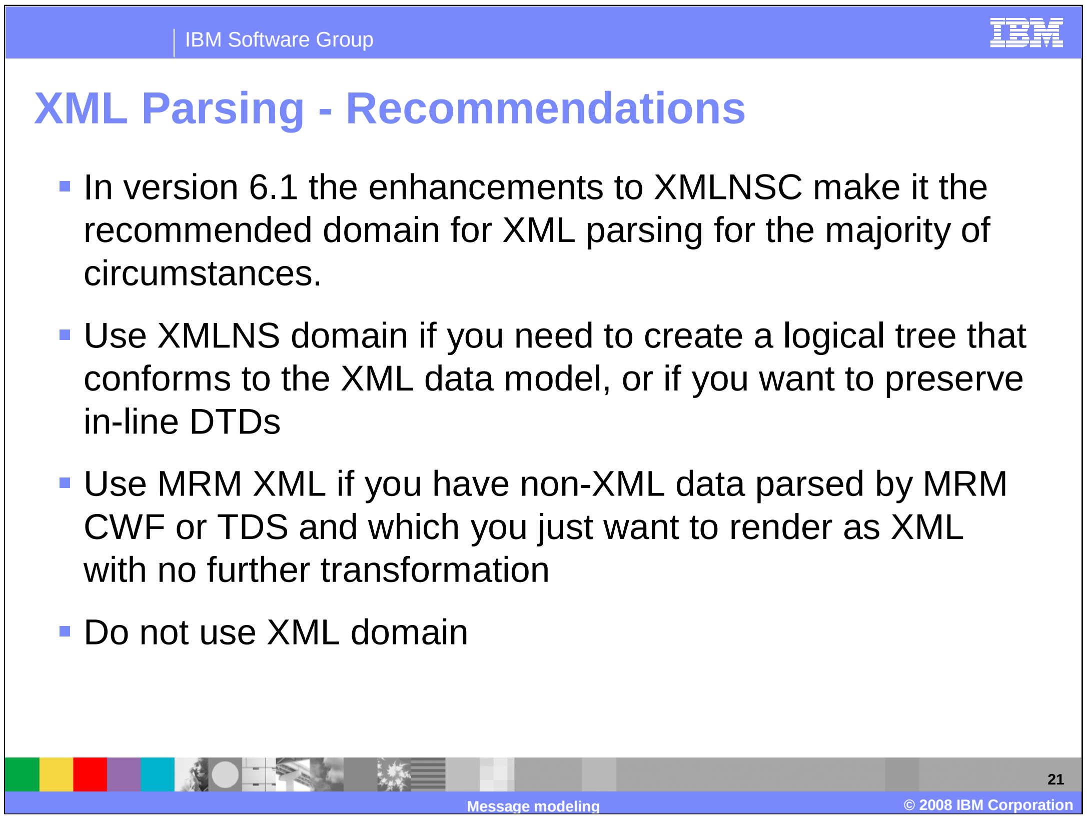
If not validating, can parse named elements 'opaquely'

20

Message modeling

© 2008 IBM Corporation

These screen captures illustrate how all these new XMLNSC capabilities can be configured on a node.


IBM Software Group

XML Parsing - Recommendations

- In version 6.1 the enhancements to XMLNSC make it the recommended domain for XML parsing for the majority of circumstances.
- Use XMLNS domain if you need to create a logical tree that conforms to the XML data model, or if you want to preserve in-line DTDs
- Use MRM XML if you have non-XML data parsed by MRM CWF or TDS and which you just want to render as XML with no further transformation
- Do not use XML domain

21

Message modeling

© 2008 IBM Corporation

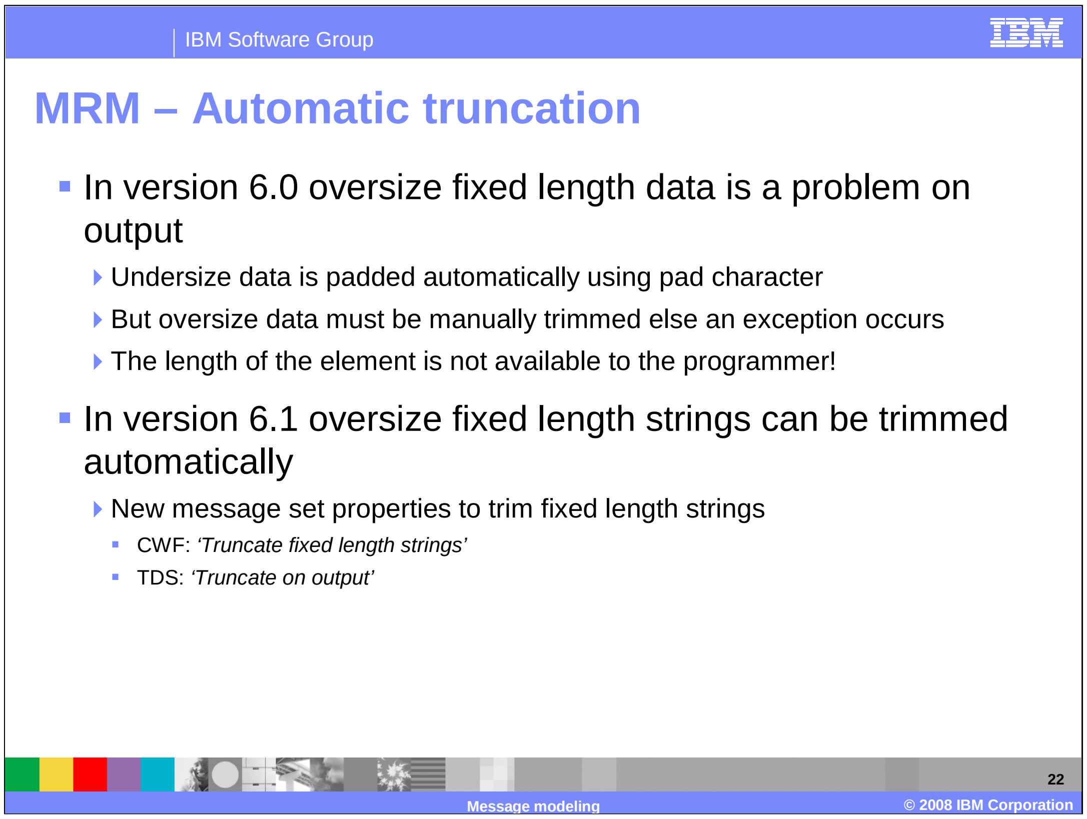
This slide summarizes the recommendations for XML parsing in Message Broker version 6.1.

First, for all new applications, you should use the XMLNSC parser if possible.

Second, the XMLNS domain can be used to create a logical tree conforming to the XML data model. It can also be used to preserve inline DTDs.

The MRM domain can be used for non-XML data, where you want to change the data to an XML format, but perform no further processing.

The XML domain is deprecated, and should not be used for new applications.

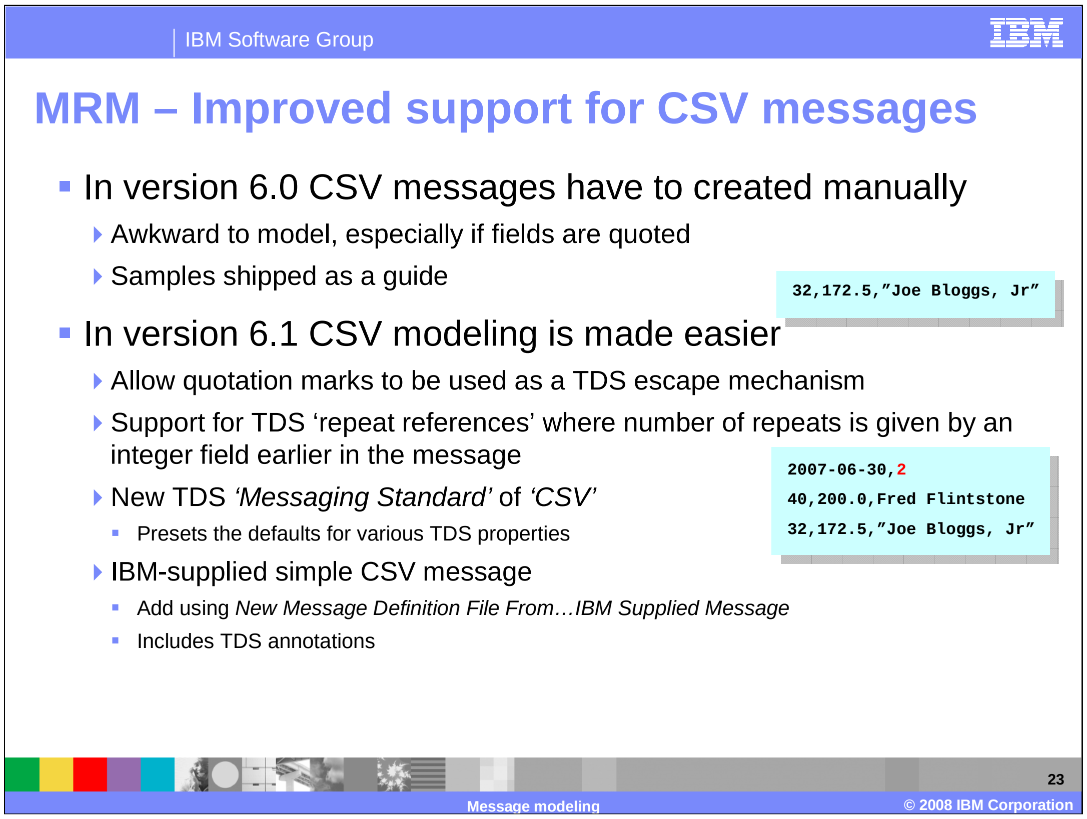


MRM – Automatic truncation

- In version 6.0 oversize fixed length data is a problem on output
  - Undersize data is padded automatically using pad character
  - But oversize data must be manually trimmed else an exception occurs
  - The length of the element is not available to the programmer!
- In version 6.1 oversize fixed length strings can be trimmed automatically
  - New message set properties to trim fixed length strings
    - CWF: *'Truncate fixed length strings'*
    - TDS: *'Truncate on output'*

This slide discusses automatic truncation of fixed length data. When you write flow logic to create output data destined to be written out as a CWF or TDS fixed length field, then if the data is too long it will cause an exception to be thrown. If you are happy for the data to be truncated to fit the field, you must code additional logic to do so. Further, this must use hard-coded lengths as there is no API available to access the model to obtain the length.

In version 6.1, new CWF and TDS options are available to truncate oversize output data before being written. These options are message set properties, CWF *'Truncate fixed length strings'* and TDS *'Truncate on output'*. They apply to all fixed length fields in all message definition fields in the message set.



MRM – Improved support for CSV messages

- In version 6.0 CSV messages have to created manually
  - Awkward to model, especially if fields are quoted
  - Samples shipped as a guide

```
32,172.5,"Joe Bloggs, Jr"
```

- In version 6.1 CSV modeling is made easier
  - Allow quotation marks to be used as a TDS escape mechanism
  - Support for TDS 'repeat references' where number of repeats is given by an integer field earlier in the message
  - New TDS *'Messaging Standard'* of *'CSV'*
    - Presets the defaults for various TDS properties
  - IBM-supplied simple CSV message
    - Add using *New Message Definition File From…IBM Supplied Message*
    - Includes TDS annotations

```
2007-06-30,2
40,200.0,Fred Flintstone
32,172.5,"Joe Bloggs, Jr"
```

Comma Separated Values, or CSV messages are very common and have been supported by TDS for several releases. To make modeling CSV messages easier, several features have been added to the TDS parser in version 6.1.

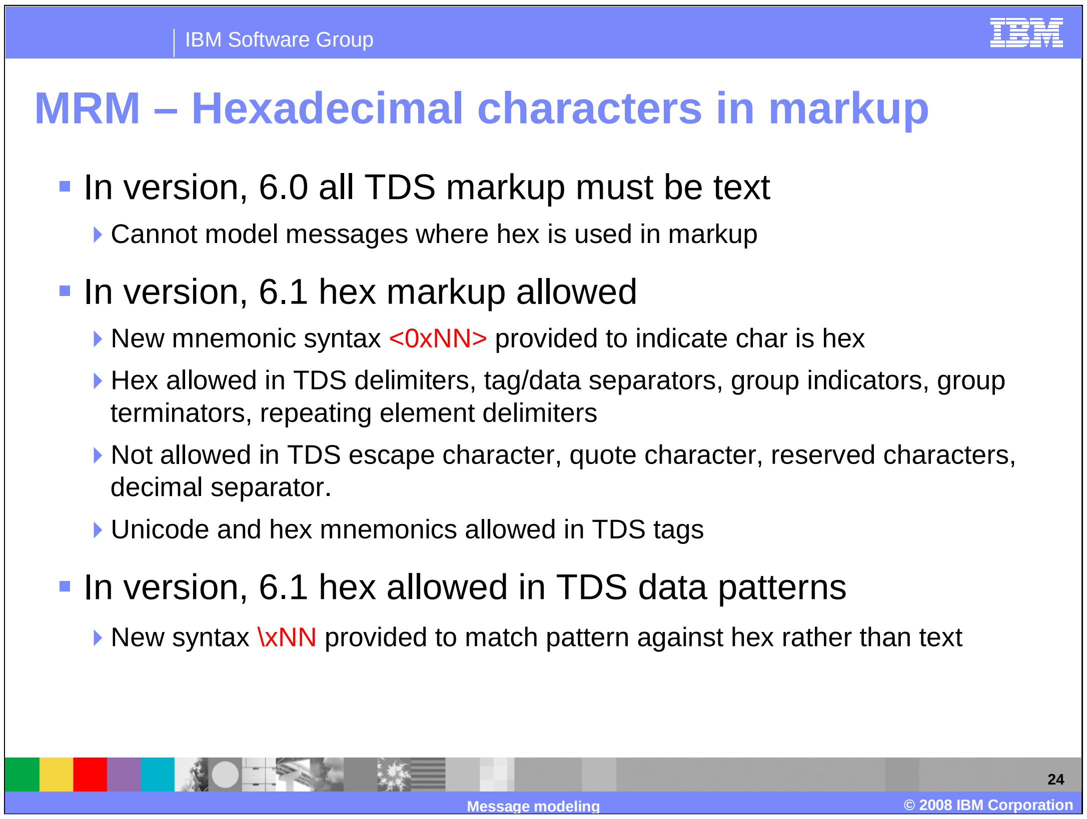
Starting and ending quotation marks can be used as an escape mechanism. A new TDS message set property *'Quote Character'* is provided. When parsing, quotation marks cause all enclosed characters to be treated as data and not potentially as markup, and the quotation marks are always stripped before data is added to the tree. On output quotation marks are added when certain characters are found in the data, the characters being those listed in the TDS *'Reserved Characters'* property.

It is now possible to allow the number of repeats of an element to be indicated by an integer field in the message. A new TDS element property *'Repeat Reference'* is provided, which works like the existing CWF property of the same name.

A new setting call 'CSV' is available, which presets the values of TDS properties like *'Data Element Separation'*, *'Quote Character'* and *'Delimiter'* to values suitable for CSV messages.

Finally, a pre-built TDS model of a simple CSV message is available. This can be added to a message set using the "*New Message Definition File From … IBM Supplied Message"* wizard.

Note that quotation mark escaping and repeat reference support can be used when modeling any message format, not just CSV.



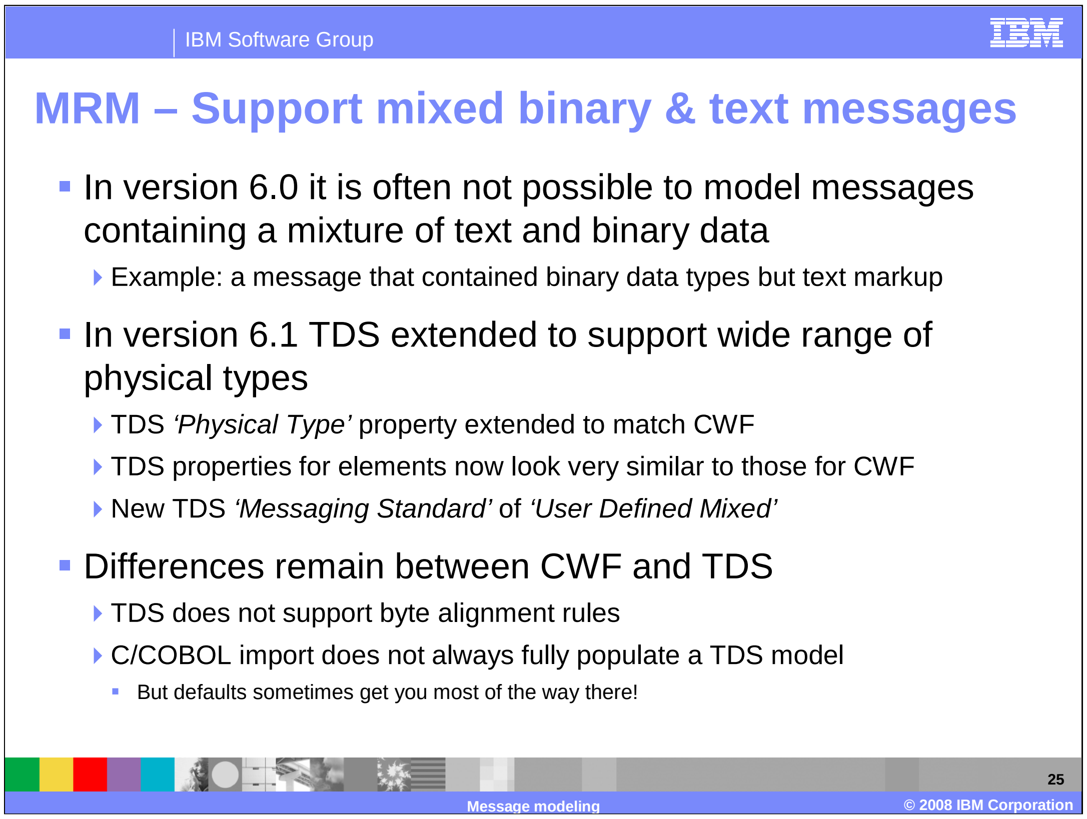
MRM – Hexadecimal characters in markup

- In version, 6.0 all TDS markup must be text
  - Cannot model messages where hex is used in markup
- In version, 6.1 hex markup allowed
  - New mnemonic syntax <0xNN> provided to indicate char is hex
  - Hex allowed in TDS delimiters, tag/data separators, group indicators, group terminators, repeating element delimiters
  - Not allowed in TDS escape character, quote character, reserved characters, decimal separator.
  - Unicode and hex mnemonics allowed in TDS tags
- In version, 6.1 hex allowed in TDS data patterns
  - New syntax \xNN provided to match pattern against hex rather than text

This slide discusses Hexadecimal characters in TDS markup. “Markup” is the term used to describe those characters in a message format that are not data values. TDS delimiters, tag/data separators, group indicators, group terminators and repeating element delimiters are all examples of markup. In version 6.1, hex characters, in addition to text characters, are permitted to occur in the markup listed. The syntax for specifying a single hex character is mnemonic “0xNN”, as shown on this slide, where NN is a hex character in the range 00 to FF. Such hex characters are not subject to code page conversion.

TDS tags can now contain Unicode character “U+NNNN” and hex character “0xNN” mnemonics. In earlier versions of Message Broker, mnemonics were not been allowed in tags.

Hex characters are now permitted to occur in TDS data patterns, used when parsing using regular expressions. The syntax for specifying a single hex character is \xNN where NN is a hex char in the range 00 to FF. Such hex characters are not subject to code page conversion during matching of the regular expression.



MRM – Support mixed binary & text messages

- In version 6.0 it is often not possible to model messages containing a mixture of text and binary data
  - Example: a message that contained binary data types but text markup
- In version 6.1 TDS extended to support wide range of physical types
  - TDS *'Physical Type'* property extended to match CWF
  - TDS properties for elements now look very similar to those for CWF
  - New TDS *'Messaging Standard'* of *'User Defined Mixed'*
- Differences remain between CWF and TDS
  - TDS does not support byte alignment rules
  - C/COBOL import does not always fully populate a TDS model
    - But defaults sometimes get you most of the way there!

Because of the historical split of the MRM's non-XML model into binary and text physical formats, it is often not possible to model message formats containing a mixture of binary data and text markup. Most kinds of binary data are only handled by CWF, while markup is only handled by TDS. In version 6.1 this changes, as the TDS parser is extended to handle a wider range of binary data types.

Key to this change is to add more capability to existing TDS element property *'Physical Type'*. The existing settings *'Characters'* and *'Messaging Standard Alternate'* have been renamed *'Text'* and *'TLOG Specific'*. New general purpose settings have been added, which behave just like their CWF equivalents. These new settings are *'Null Terminated String', 'Length Encoded String 1', 'Length Encoded String 2', 'Packed Decimal', 'External Decimal', 'Integer', 'Float', 'Time Seconds', 'Time Milliseconds'* and *'Binary'* .

To go along with the new physical types, extra TDS element properties such as *'Length Units'* have been added, while others such as *'Sign Orientation'* have been extended. There are also new TDS message set properties to match CWF equivalents.

The net result is that TDS matches CWF except for two things:

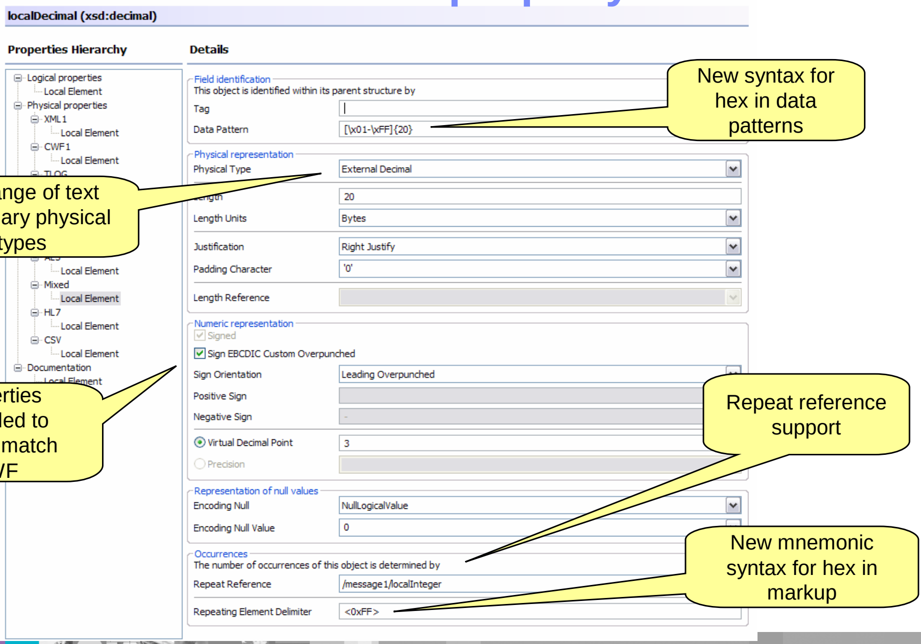
TDS does not support byte alignment rules. There are no equivalent to CWF element properties *'Byte Alignment', 'Leading Skip Count',* and *'Trailing Skip Count'*.

The C and COBOL importers do not always fully populate a TDS model. These importers set logical properties and CWF physical format properties, but not TDS physical format properties. However, in conjunction with new TDS *'Messaging Standard'* setting *'User Defined Mixed'*, the TDS physical format derives many defaults from the logical model. These include TDS *'Physical Type', 'Justification', 'Length'* and *'Sign Orientation'*, which are sometimes enough to fully model the message.

Therefore, use CWF for pure COBOL and C modeling.

IBM Software Group

MRM – New TDS Element property sheet


New syntax for hex in data patterns
Full range of text and binary physical types
Properties extended to closely match CWF
Repeat reference support
New mnemonic syntax for hex in markup

localDecimal (xsd:decimal)

Properties Hierarchy

- Logical properties
  - Local Element
- Physical properties
  - XML1
    - Local Element
  - CWF1
    - Local Element
  - TLOG
  - ...
    - Local Element
  - Mixed
    - Local Element
  - HL7
    - Local Element
  - CSV
    - Local Element
- Documentation
  - Local Element

Details

Field identification
This object is identified within its parent structure by

Tag	
Data Pattern	[\x01-\xFF]{20}

Physical representation

Physical Type	External Decimal
Length	20
Length Units	Bytes
Justification	Right Justify
Padding Character	'0'
Length Reference	

Numeric representation
✓ Signed
✓ Sign EBCDIC Custom Overpunched

Sign Orientation	Leading Overpunched
Positive Sign	
Negative Sign	-
● Virtual Decimal Point	3
○ Precision	

Representation of null values

Encoding Null	NullLogicalValue
Encoding Null Value	0

Occurrences
The number of occurrences of this object is determined by

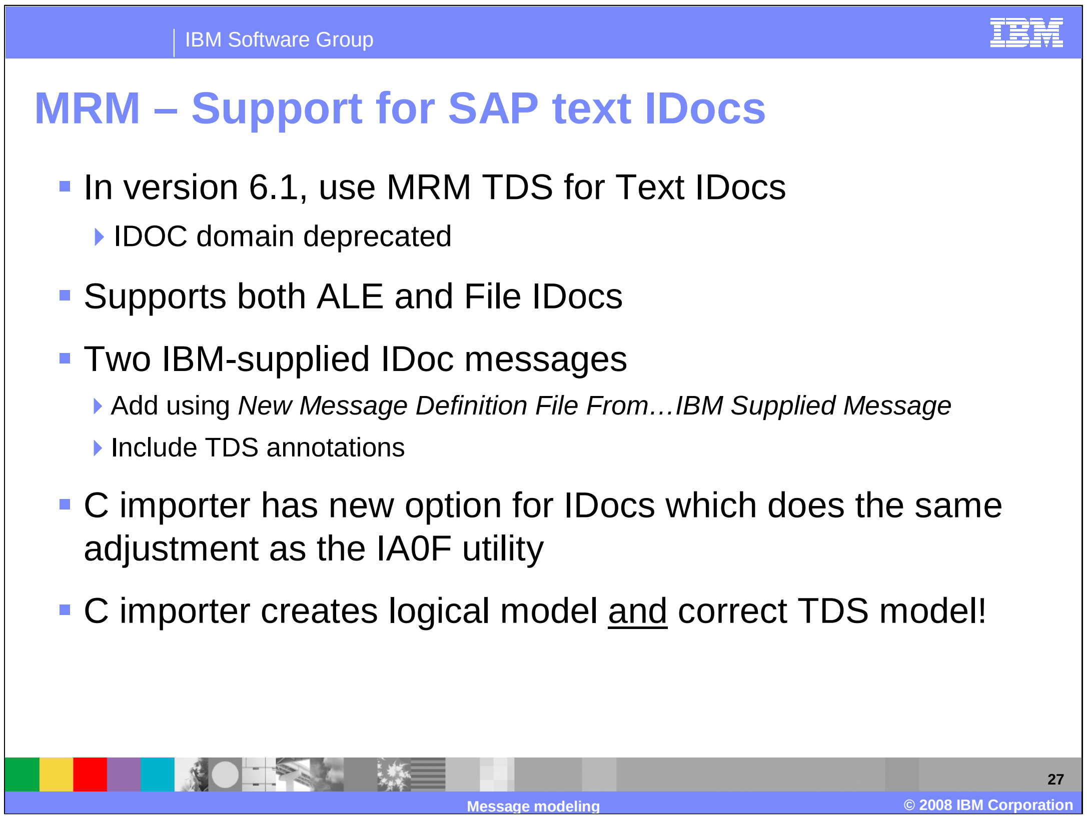
Repeat Reference	/message1/localInteger
Repeating Element Delimiter	<0xFF>

26

Message modeling

© 2008 IBM Corporation

Some of these properties are shown on this screen capture.


IBM Software Group

MRM – Support for SAP text IDocs

- In version 6.1, use MRM TDS for Text IDocs
  - IDOC domain deprecated
- Supports both ALE and File IDocs
- Two IBM-supplied IDoc messages
  - Add using *New Message Definition File From…IBM Supplied Message*
  - Include TDS annotations
- C importer has new option for IDocs which does the same adjustment as the IA0F utility
- C importer creates logical model and correct TDS model!

27

Message modeling © 2008 IBM Corporation

The IDOC domain has some limitations, the main one being that it is incompatible with the Mapping node. The import step is not straight-forward, and requires some pre-processing and post-processing of the C header files using a utility program. Finally, while ALE IDocs are supported, File IDocs, which have DC and DDs separated by a Line-Feed, are not.

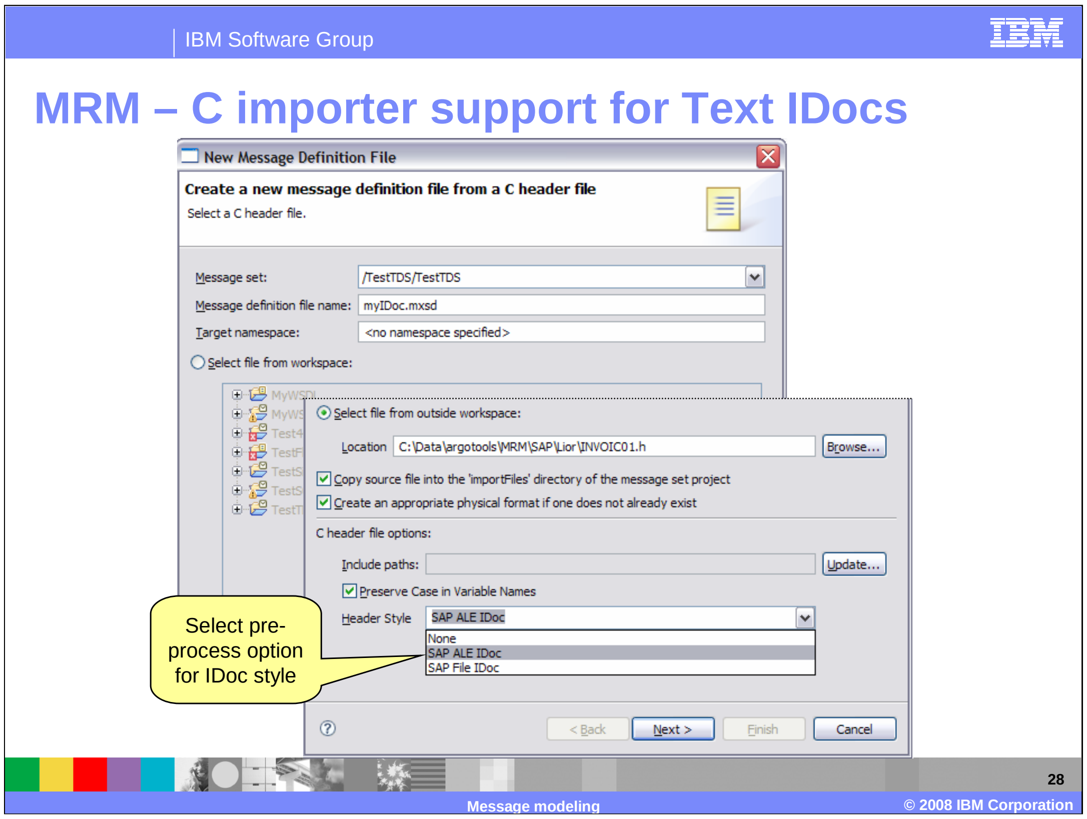
Rather than enhance the IDOC parser to cope with these problems, in version 6.1 the *MRM TDS* parser now has all the capability to parse both ALE and File IDocs. So in version 6.1, text IDocs should be parsed using MRM TDS, and the IDOC domain is deprecated. The procedure for modeling such IDocs has these steps:

First, import a pre-built TDS model of an IDoc into your message set using the "*New Message Definition File From … IBM Supplied Message"* wizard. There are pre-built models for ALE and File flavors. This will create a TDS physical format called *'Text_IDoc'* if none exists.

Second, ensure your TDS physical format has the *'Messaging Standard'* property set to *'User Defined Text'.*

Finally, import the user structure C header files using the "*New Message Definition File From … C"* wizard, and select the new option for ALE or File IDocs. No pre-processing is needed, as the utility logic is now in the wizard. The user structures are linked to the DD using MRM multipart messaging.

Conveniently, the default TDS model created by the C importer correctly models the user structure without any manual editing, because IDoc data is all fixed length strings.



MRM – C importer support for Text IDocs

New Message Definition File

Create a new message definition file from a C header file

Select a C header file.

Message set: /TestTDS/TestTDS

Message definition file name: myIDoc.mxsd

Target namespace: <no namespace specified>

Select file from workspace:

[✓ Select file from outside workspace:]

Location C:\Data\argotools\MRM\SAP\Lior\INVOIC01.h

✓ Copy source file into the 'importFiles' directory of the message set project

✓ Create an appropriate physical format if one does not already exist

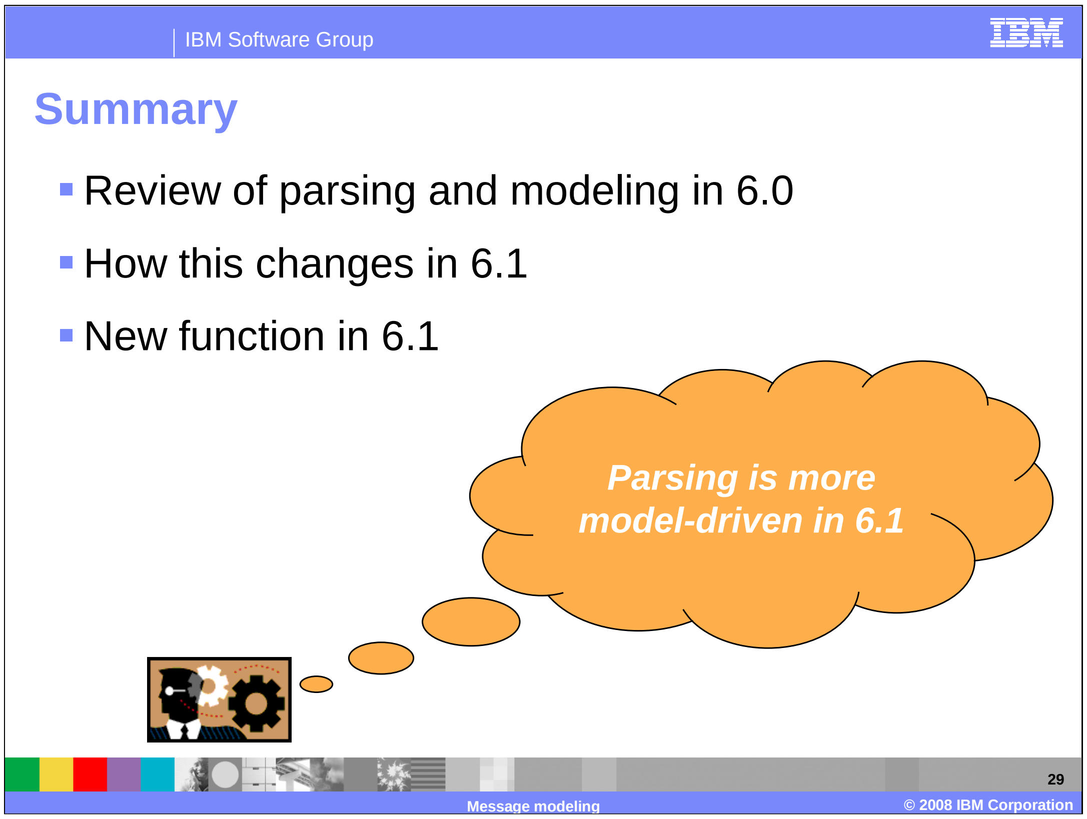
C header file options:

Include paths:

✓ Preserve Case in Variable Names

Header Style: SAP ALE IDoc

This screen capture shows the process for creating a message definition from a text IDoc.

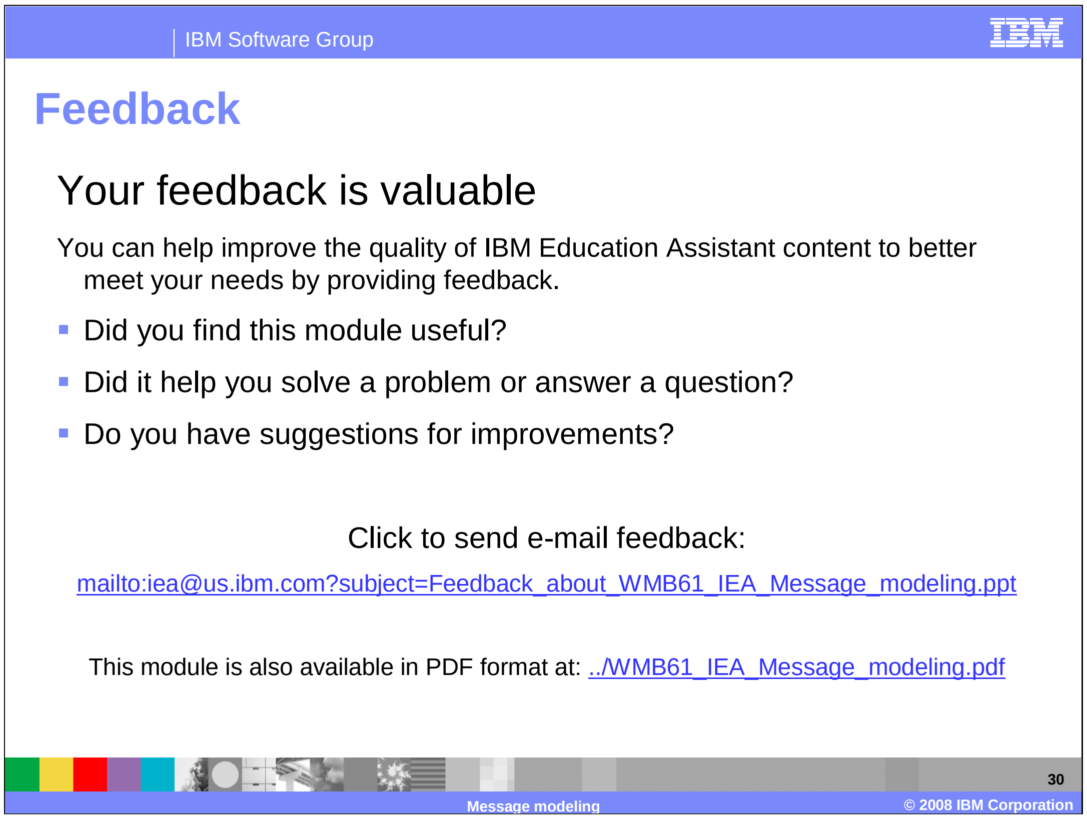


Summary

- Review of parsing and modeling in 6.0
- How this changes in 6.1
- New function in 6.1

In summary, this presentation provided a review of message modeling techniques in Message Broker version 6.0.

It then discussed how these have changed in version 6.1, and finally discussed the new function in version 6.1 to assist with message modeling.



Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

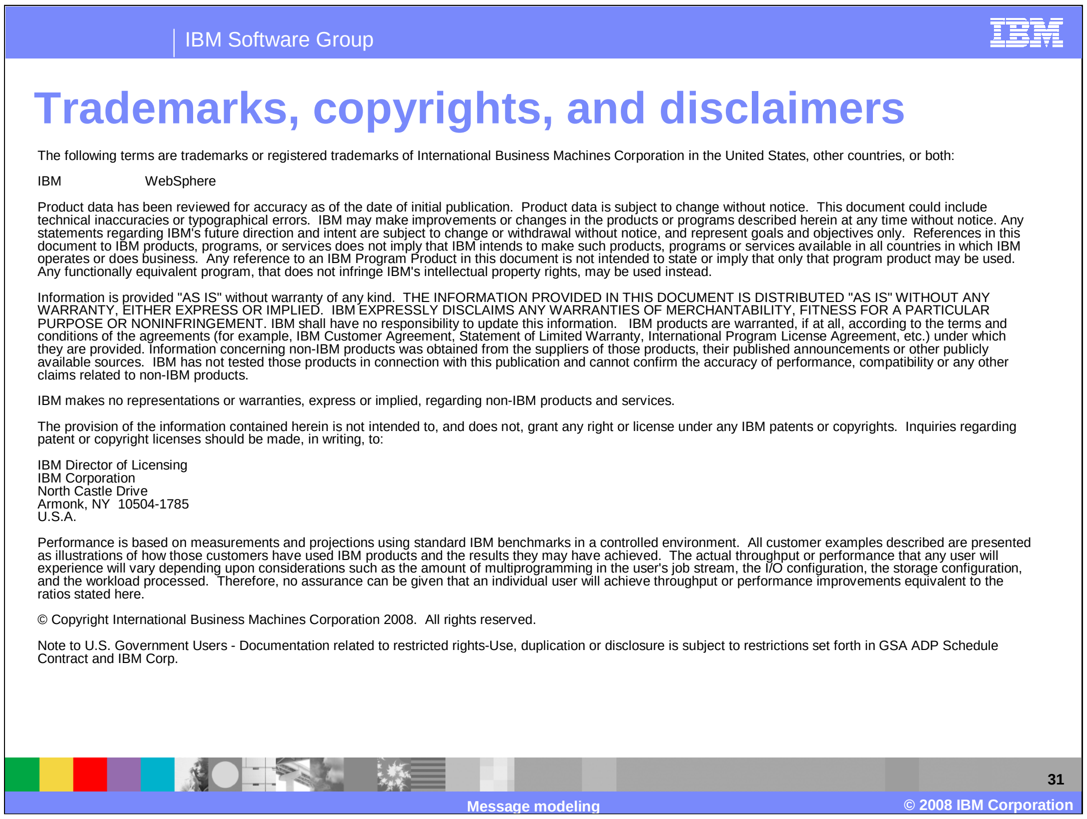
- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WMB61_IEA_Message_modeling.ppt

This module is also available in PDF format at: ../WMB61_IEA_Message_modeling.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.



Trademarks, copyrights, and disclaimers