Message modeling and parsing enhancements in WebSphere Message Broker V6.1

Steve M Hanson
Sanjay Nagchowdhury

October 15, 2008

Message modeling lets you predefine message formats so that WebSphere Message Broker can then automatically parse and write them. This article shows you how to use the enhanced message modeling and parsing in Message Broker V6.1.

Introduction

IBM® 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. Each parser is suited to a particular class of messages (such as fixed-length binary, delimited text, or XML) known as a message domain. For architects, message set designers, and developers, this article shows you how to use the enhancements in message modeling and parsing in WebSphere Message Broker V6.1 for the XMLNSC and MRM domains.

What is message parsing?

Typically a WebSphere Message Broker message flow receives messages in a defined format, transforms them, and outputs them in a different format. Figure 1 below shows an example where a COBOL Person data structure is transformed into an XML Person document. Each 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 and transformed by the message flow, it must be converted into a logical message tree, which 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 that transforms the message and creates a new, output logical message tree, which is converted into an output bit stream that represents the message as an XML document.

The component of the broker that converts a bit stream into a logical message tree (parsing), and a logical message tree back into a bit stream (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.

Only the parser 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 of using a model later in the article.

**Types of parsers**

Obviously, a parser must understand the format of the messages that it will be parsing and serializing. There are two basic approaches to writing a parser:
• Programmatic approach -- A specific parser program is written to parse each message format. For example, if you have five different COBOL data structures, you write five format-specific parsers, with the format information hard-wired into the parser program code. This approach works well when the format is fixed or self-defining, and it has the advantage of facilitating easy optimizations for specific formats. The disadvantage is lack of flexibility -- you must write a new parser program for each new format, and if a format changes, you must change, recompile, relink, and redeploy the parser program.

• Descriptive approach -- You write a single general-purpose model-driven parser program to parse all formats, with each message format represented by its own model. The knowledge of the format is not in the parser but in the model, which is either accessed directly by the parser at run time, or generated into code that the parser invokes. Model-driven parsing is a flexible approach because it is easier to change and deploy models, but a disadvantage is that optimization is more difficult with a general-purpose parser.

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

Why model messages?
There are several reasons to model your messages. The following reasons apply to use of a model by a parser at runtime:

• Most messages are not self-defining -- Often, a model is needed to parse the message bit stream. For example, a message created by a C or COBOL program is just a stream of binary data, meaningless without some intelligence to interpret it, and that's where the model comes in. XML is the opposite -- it contains embedded structural information that enables an XML parser to parse any XML document without using a model.

• Validation requires a model -- 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 still needs a model to perform validation.

The following reasons apply to using a model at design time. They are independent of whether the model is actually used by a parser.

• Speeds transformation development -- 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.

• Provides version control -- When stored in a central shared repository, models provide a good way to track different versions of your messages.

• Provides documentation -- Models provide message documentation that can be shared among programmers, business analysts, and integration specialists.

Parsing, modeling, and domains in WebSphere Message Broker V6
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, and some domains support several different
classes of message. The domain 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. For example:

- For programmatic parsing of XML documents without a model, there are a set of generic XML domains: XML, XMLNS, and XMLNSC.
- The MRM domain is the most flexible domain. It has a general-purpose, model-driven parser that can parse binary messages, text messages, and XML documents.
- Specialist domains are provided for MIME messages (SwA, RosettaNet, and so on) and for SAP IDOC structures.
- The BLOB domain can be used when the message is treated as opaque, not parsed, and simply routed.

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.

**Generic XML domains in WebSphere Message Broker V6**

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

- **XML domain** -- Original way of processing XML documents. It has no support for XML namespaces and is not recommended for developing new message flows.
- **XMLNS domain** -- Added to provide 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, so formatting white space is retained and simple elements and attributes have their value held in a separate child node of the tree.
- **XMLNSC compact domain** -- Added in WebSphere Message Broker V6.0 to reduce the amount of memory occupied by logical message trees created from XML documents. This compact behaviour 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. A big memory saving can result. The XMLNSC domain is the first domain to exploit a facility 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 behaviour to be controlled on a per node basis.

**MRM domain in WebSphere Message Broker V6.0**

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 (which are available as an extension), 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). When parsing XML, the MRM behaves as a 'compact' domain like XMLNSC.

**IDOC domain in WebSphere Message Broker V6.0**

In WebSphere Message Broker V6.0, the IDOC domain is the recommended way for parsing SAP Application Link Enabling (ALE) text IDocs delivered to broker from the WebSphere MQ Link for R/3. However the IDOC domain has some limitations:

- It is incompatible with the Mapping node because an MRM element is inserted in the path (for example, Root.DD[1].MRM.xxx).
- The import step requires both pre-processing (for example, removal of #define, padding to 1000 bytes) and post-processing (upper-casing of message names) of the C header files using IA0F utility programs. The [IA0F Support Pac](https://www.ibm.com/support/docview.wss?uid=swg21647972) contains a set of tools to help you create message sets when using the IDOC parser.
- File IDocs are not supported.

**Creating and using message sets in WebSphere Message Broker V6.0**

Message models live in a container called a message set. Typical sequence of events when creating and using a message set:

1. Create a message set project and message set.
2. 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.
3. Alternatively, you can create an empty message definition file and create your logical structure and physical annotations using just the editor.
4. 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.

5. 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.

6. 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.

Figure 2 shows the sequence of events when a message set is deployed to the Message Broker runtime in WebSphere Message Broker V6.0.

Figure 2. Deploying a message set to the runtime in WebSphere Message Broker V6.0

Parsing and modeling enhancements in WebSphere Message Broker V6.1

We will now describe the enhancements for the parsers and domains used in WebSphere Message Broker V6.1.

Two key existing domains are enhanced:

- **XMLNSC domain** -- Now uses a new internal high-speed parser. The XMLNSC domain can be used without a model as in WebSphere Message Broker V6.0, but can also parse and validate against a real XML Schema, so becoming model-driven.

- **MRM domain** Enhanced to increase the number of non-XML message formats that can be modeled and to make modeling easier.
Two new model-driven domains are added for WebSphere Message Broker V6.1:

- **SOAP domain** -- For use with the new SOAP nodes provided in WebSphere Message Broker V6.1. These nodes handle SOAP XML, MIME wrappered SOAP with Attachments and MTOM, nd provide support for WS-Addressing and WS-Security standards. A canonical tree is created, which is owned by this new domain.

- **DataObject domain** -- For use with the new 6.1 Adapter nodes. These nodes interface directly with EIS systems using embedded WebSphere adapters. A tree is created to represent the adapter business object, which is owned by this new domain.

Two existing domains are officially deprecated:

- **XML domain** -- No longer needed because XMLNSC or XMLNS can be used instead.

- **IDOC domain** -- Limited function. MRM can now handle SAP ALE and File IDocs and should be used instead.

In this case, deprecation has the meaning that it will not be enhanced and will be withdrawn in a future version of WMB.

**Changes to message sets in WebSphere Message Broker V6.1**

There are some key changes to the toolkit in WebSphere Message Broker V6.1 which affect how a model is deployed to the message broker runtime. In WebSphere Message Broker V6.1, a new property is introduced called 'Supported message domains'. 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 the MRM and IDOC domains, a message dictionary is required at runtime. A .dictionary file is generated from the message set and added to the .bar file. This is the same behaviour as WebSphere Message Broker V6.0.

At runtime, the XMLNSC domain requires XML Schema if validating, the DataObject domain requires XML Schema and the SOAP domain requires XML Schema and WSDL. For these domains an .xsd file is generated from each message definition file. The .xsd files are zipped into a single .xsdzip file. The .xsdzip file is added to the .bar file. Additionally for SOAP, all .wsdl files are added to the .xsdzip file.

Therefore, depending on the supported domains, a message set can cause a .dictionary file and/or .xsdzip file to be generated and added to the .bar file. Changes to message sets in WebSphere Message Broker V6.1:
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, an .xsdzip file is created to hold the generated XML Schema and the original WSDL files. This is unzipped 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. Figure 4 shows the new sequence of events when a message set is deployed to the Message Broker runtime in WebSphere Message Broker V6.1:
Figure 4. Deploying a message set to the runtime in WebSphere Message Broker V6.1

Enhancements to the XMLNSC Domain

There have been several important changes to the way that the XMLNSC domain operates. The enhancements to XMLNSC make it the recommended domain for XML parsing for the majority of circumstances.

New internal high-speed parser

A new, faster XML parser has been developed by IBM Research. It is a direct functional replacement for IBM's Xerces parser, so the XMLNSC domain has switched to use the new parser with consequent improvement in performance. This switch has no external impact. A WebSphere Message Broker V6.0 message flow that specifies XMLNSC will operate unchanged when migrated to WebSphere Message Broker V6.1. Note that this change has only been made to XMLNSC, not to XML, XMLNS or MRM XML which continue to use Xerces.

The architecture of the XMLNSC domain means that it uses significantly less CPU than the other XML domains. XMLNSC also uses less memory when building a message tree from an XML document. Therefore, the XMLNSC domain should be used if performance is critical to your application.

Validation against XML Schema

The validation logic used by the MRM parser does not conform exactly to the W3C XML Schema specification. The XMLNSC parser in WebSphere Message Broker v6.1 addresses this shortcoming, and provides standards-compliant schema validation.
XMLNSC validation considerably out-performs MRM validation too, as the architecture of the new XML parser is optimized for the model-driven scenario. Validation is fully compliant with the ML Schema 1.0 specification.

XMLNSC operates in one of two modes. The default for WebSphere Message Broker V6.0 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.

Whether to validate is controlled by the existing node Validate option, just like MRM. If the Validate option is set to None, then the message will not be validated. If the Validate option is set to Content and value or Content, then the message will be validated against an XML Schema. Figure 5 shows how to switch on validation in order to parse and validate against an XML Schema:

**Figure 5. Switch on validation for the XMLNSC domain**

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. (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). Figure 6 shows how to supply the Message Set property when validation is selected:
Correct data types in tree

In WebSphere Message Broker V6.0, 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 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 MRM builds its tree. The default for WebSphere Message Broker V6.0 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 may be found under XMLNSC Parser Options and is only enabled in validating mode. Figure 7 shows how to select the new option Build tree using XML Schema data types.

Opaque parsing

This is a term used to describe a performance enhancement, where named XML elements are not fully parsed, but instead are skipped and the raw bitstream inserted into the tree as a Unicode string. It is available in WebSphere Message Broker V6.0 but only exposed in a limited way. In WebSphere Message Broker V6.1 it’s been fully externalised 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 may be found under XMLNSC Parser Options and is only enabled in non-validating mode, because by definition validation involves fully expanding all elements. Figure 7 shows how to specify the XPaths of the elements you want to parse opaquely in the new option Opaque elements:
The enhancements to XMLNSC make it the recommended domain for XML parsing for the majority of circumstances. The XMLNS domain should be used if you need to create a logical tree that conforms more closely to the XML infoset (XPath), or if you want to preserve in-line DTDs. The MRM-XML domain should only be used for new message flows if you have non-XML data parsed by MRM CWF or TDS which you just want to render as XML with no further transformation. The XML domain has been officially deprecated and should not be used for new applications.

**Enhancements to the MRM Domain**

There have been several improvements to the MRM domain for binary and text messages.

**Automatic truncation of fixed length data**

If a message flow in WebSphere Message Broker V6.0 creates output data destined to be written out as a CWF or TDS fixed length field and if the data is too long, then an exception is thrown. If you want to avoid the exception, then the data must be truncated to fit the field and you must code additional logic to do the truncation. Also, this must use hard-coded lengths as there is no API available to access the model to obtain the length.

In WebSphere Message Broker V6.1 new CWF and TDS options are available to truncate oversize output data before being written. They apply to all fixed length fields in all message definition files in the message set. These new CWF and TDS options are shown in Figures 8 and 9:
Hexadecimal values 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 WebSphere Message Broker V6.0, it is not possible to model messages where hexadecimal values are used in TDS markup or TDS tags. The following enhancements have been made to WebSphere Message Broker V6.1:

- TDS markup can contain hexadecimal values in addition to the more usual text characters. The syntax for specifying a single hexadecimal value is mnemonic <0xNN> where NN is a
hexadecimal digit in the range 00 to FF. These numbers represent a Unicode character, not a character in the code page of the input message.

• TDS tags can now contain unicode characters and hexadecimal character <0xNN> mnemonics.

• TDS data patterns can now contain hexadecimal values. This allows hexadecimal values in regular expressions when parsing. A hexadecimal value is specified as \xNN, where N is a hexadecimal digit in the range 0 to F.

**Improved support for CSV messages**

Comma Separated Values (CSV) messages are very common and have been supported by TDS for several releases. To make modeling CSV messages easier, the following enhancements have been added to TDS in WebSphere Message Broker V6.1:

• A new TDS message set property Quote Character is provided. This allows starting and ending quote characters to be used as an escape mechanism. When parsing, quotes cause all enclosed characters to be treated as data and not potentially as markup, and the quotes are always stripped before data is added to the tree. On output, any data field containing one of the characters specified in the TDS Reserved Characters message set property will have quotes added (This is analogous to the existing behaviour for the TDS Escape character message set property).

• A new TDS element property Repeat Reference is provided. This allows the number of repeats of an element to be indicated by an integer field in the message. his works in the same way as the existing CWF property of the same name.

• Provide a new TDS Messaging Standard setting called CSV, which presets the values of TDS properties like Data Element Separation, Quote Character, Reserved Characters and Delimiter to values suitable for CSV.

• Provide a pre-built TDS model of a simple CSV message that can be added to a message set using the New Message Definition File From ... IBM Supplied Message wizard.

The new TDS message set property Quote Character and new TDS Messaging Standard setting called CSV are shown in Figure 10:
Improvements to model messages containing text and binary data

In WebSphere Message Broker V6.0, most kinds of binary data are only handled by CWF, while markup is only handled by TDS. In WebSphere Message Broker V6.1, TDS is extended to handle a wider range of binary data types. This means that TDS element properties now look very similar to CWF element properties. Key to this change is to add more capability to the 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 Null Terminated String, Length Encoded String 1, Length Encoded String 2, Packed Decimal, External Decimal, Integer, Float, Time Seconds, Time Milliseconds and Binary have been added. These new settings behave just like their CWF equivalents.

The following new TDS element properties have been added in WebSphere Message Broker V6.1 for use with the new physical types:

- Length Units
- Signed
- Sign EBCDIC Custom Overpunched

These new TDS element properties are shown in Figure 11:
In WebSphere Message Broker V6.1, some TDS element properties have been extended. Table 1 shows the updates for these TDS element properties:

**Table 1. TDS element properties which have been extended**

<table>
<thead>
<tr>
<th>TDS element property</th>
<th>WebSphere Message Broker V6.0 setting</th>
<th>WebSphere Message Broker V6.1 setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign Orientation</td>
<td>None</td>
<td>Leading Separate</td>
</tr>
<tr>
<td></td>
<td>Leading</td>
<td>Trailing Separate</td>
</tr>
<tr>
<td></td>
<td>Trailing</td>
<td>Leading Overpunched</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trailing Overpunched</td>
</tr>
<tr>
<td>Encoding Null</td>
<td>NullPadFill</td>
<td>NullPadFill</td>
</tr>
<tr>
<td></td>
<td>NullLogicalValue</td>
<td>NullLogicalValue</td>
</tr>
<tr>
<td></td>
<td>NullLiteralValue</td>
<td>NullLiteralValue</td>
</tr>
<tr>
<td></td>
<td>NullLiteralFill</td>
<td>NullLiteralFill</td>
</tr>
</tbody>
</table>

The Sign Orientation 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. These updated TDS element properties are shown in Figure 12:

**Figure 12. Updated TDS element properties**

![Figure 12. Updated TDS element properties](image)

In WebSphere Message Broker V6.1, new TDS message set properties have been added to match the CWF equivalents. These new properties are listed in table 2.

**Table 2. New TDS message set properties**

<table>
<thead>
<tr>
<th>Message Set Property Setting</th>
<th>New Message Set Property in WebSphere Message Broker V6.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messaging Standard</td>
<td>User Defined Mixed - for mixed text and binary data</td>
</tr>
<tr>
<td>Numeric settings</td>
<td>Packed decimal positive code</td>
</tr>
<tr>
<td></td>
<td>Derive sign from logical type</td>
</tr>
<tr>
<td></td>
<td>Default byte order</td>
</tr>
<tr>
<td></td>
<td>Default packed decimal byte order</td>
</tr>
<tr>
<td></td>
<td>Default float format</td>
</tr>
<tr>
<td>Representation of boolean values</td>
<td>Binary boolean true value</td>
</tr>
<tr>
<td></td>
<td>Binary boolean false value</td>
</tr>
<tr>
<td>General settings</td>
<td>Output policy for missing elements</td>
</tr>
</tbody>
</table>

These new TDS message set properties are shown in Figure 13:
These extensions to TDS make TDS almost match CWF apart from the following differences:

- TDS does not support byte alignment rules. There are no equivalents to CWF element properties Byte Alignment, Leading Skip Count, 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.

CWF should continue to be used for pure COBOL and C modeling.

**Support for SAP text IDocs**

MRM TDS now has all the capability to parse both ALE and File IDocs. In WebSphere Message Broker V6.1, it is recommended that text IDocs sent from the WebSphere MQ Link for R/3 or exported to the file system are parsed using MRM TDS. The IDOC domain is officially deprecated. Note that, IDocs received as business objects from the new WebSphere Adapter for SAP should be handled by the new DataObject domain.

The procedure for modeling SAP text IDocs is now:

- Import a pre-built TDS model of an IDoc DC/DD into your message set using the New Message Definition File From ... IBM Supplied Message wizard. There are pre-built models for ALE and File flavours. This will create a TDS physical format called Text_IDoc if none exists.
- Ensure your TDS physical format has Messaging Standard property set to User Defined Text.
• 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 IA0F 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.

Figure 14 shows the New Message Definition File From ... C wizard which allows you to select the pre-process option for IDoc:

Figure 14. Support for SAP text IDocs

Conclusion
This article has described enhancements made to message modeling and parsing in WebSphere Message Broker V6.1. The XMLNSC domain is the recommended domain for parsing and writing XML data for the majority of applications. The XMLNSC domain has been enhanced in the following ways:

• The XMLNSC domain now uses a high performance XML parser which means that it now uses significantly less CPU than the other XML domains. XMLNSC also uses less memory when building a message tree from an XML document. This gives it significant performance improvements over the XMLNS and XML parsers. Therefore, the XMLNSC domain should be used if performance is critical to your application.
• The XMLNSC domain has become model-driven as it can now parse and validate against a real XML Schema. XMLNSC validation considerably out-performs MRM validation too, as the architecture of the new high performance XML parser is optimized for the model-driven scenario. Validation is fully compliant with the XML Schema 1.0 specification.
• The XMLNSC domain supports opaque in non-validating mode. This reduces the costs of parsing and writing the message, and might improve performance in other parts of the message flow.
The MRM domain has been enhanced in the following ways:

- Automatic truncation of fixed length data.
- Hexadecimal values can be contained in TDS markup, tags and in data patterns.
- Improved support for CSV messages, including Quote Character, Repeat Reference, new CSV Messaging Standard setting, pre-built TDS model of a simple CSV message.
- Support for mixed binary and text messages.
- Support for SAP text IDocs.
Related topics

- **MLNSC Validation sample**
  This sample is in the Message Format Samples section in Broker Technology Samples in the Toolkit Samples Gallery in WebSphere Message Broker V6.1.

- **Message Modeling**
  This topic in the WebSphere Message Broker Information Center explains message modeling, including concepts, domains, parsers, message definitions, and more.

- **WebSphere Message Broker developer resources page**
  Technical resources to help you use WebSphere Message Broker for connectivity, universal data transformation, and enterprise-level integration of disparate services, applications, and platforms to power your SOA.

- **WebSphere Message Broker product page**
  Product descriptions, product news, training information, support information, and more.

- **WebSphere Message Broker information center**
  A single Web portal to all WebSphere Message Broker V6 documentation, with conceptual, task, and reference information on installing, configuring, and using your WebSphere Message Broker environment.

- **WebSphere Message Broker documentation library**
  WebSphere Message Broker specifications and manuals.

- **WebSphere Message Broker support page**
  A searchable database of support problems and their solutions, plus downloads, fixes, problem tracking, and more.

- **Redbook: Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB**
  Patterns for e-business are a group of proven, reusable assets that can be used to increase the speed of developing and deploying e-business applications. This Redbook shows you how to use WebSphere ESB together with WebSphere Message Broker to implement an ESB within an SOA. Includes scenario to demonstrate design, development, and deployment.

- **developerWorks WebSphere Business Integration zone**
  For developers, access to WebSphere Business Integration how-to articles, downloads, tutorials, education, product info, and more.

- **WebSphere Business Integration products page**
  For both business and technical users, a handy overview of all WebSphere Business Integration products

- **Most popular WebSphere trial downloads**
  No-charge trial downloads for key WebSphere products.

- **Technical books from IBM Press**
  Convenient online ordering through Barnes & Noble.

© Copyright IBM Corporation 2008

**Trademarks**
(www.ibm.com/developerworks/ibm/trademarks/)