WebSphere MQ Integrator
MRM Utilities
Version 1.0

14th December 2001

Jim MacNair
MS 1335 Bld1
Route 100
Somers, NY 12582

macnair@us.ibm.com

Property of IBM

Take Note!

Before using this report be sure to read the general information under "Notices".

First Edition, December

This edition applies to Version 1.0 of *WebSphere MQ Integrator - MRM utilities* and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program that does not infringe any of the intellectual property rights may be used instead of the IBM product.

Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, USA.

The information contained in this document has not be submitted to any formal IBM test and is distributed AS-IS. The use of the information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While IBM has reviewed each item for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Trademarks and service marks

The following terms, used in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

- IBM
- MQSeries
- WebSphere
- WebSphere MQ Integrator
- WMQI

The following terms are trademarks of Microsoft Corporation:

- Internet Explorer
- Windows
- Windows / NT
- Windows / 2000

Acknowledgments

The author wishes to acknowledge the assistance of Sean Dunne and Mairin Whelan of the IBM Dublin lab, as well Peter Edwards in the IBM Hursley lab.

Summary of Amendments

Date	Changes
14 December 2001	Initial release

Preface

This SupportPac contains a number of command line utilities that provide useful functions when working with the MRM in WebSphere MQ Integrator V2.1. The utilities include a bulk load utility that will load type, element and value definitions into a message set that has been defined within the MRM. Utilities are also provided to list all message sets and messages defined within the MRM and to list certain aspects of the elements and types within a particular message.

Chapter 1. Overview

This SupportPac contains a number of command line utilities that can make it easier and more productive to work with WebSphere MQ Integrator V2.1 (WMQI 2.1). The following utilities are supplied with this SupportPac:

- List message sets and messages defined within the MRM (mqsilistmsgset.exe)
- Delete a particular message set
- Load definitions into a message set from a text file (mqsimrmbulkimp.exe)

All of the supplied utilities are command line driven. They all require the name of the MRM repository database plus a corresponding user id and password that have access to the MRM repository. The utilities must be executed on a system that has direct access to the MRM repository, whether the same system that the repository database resides on or a system with DB2 client access to the MRM repository database.

IBM's WebSphere MQ Integrator (WMQI) provides a powerful solution to the challenge of formatting and reformatting data. In its simplest form, WMQI takes a description of a message format (layout) and, when presented with messages in this format, can break apart that message into its constituent fields. Once the original format has been parsed, the message can be output in a different format using the fields contained in the original message as the source of input.

The new output format may simply contain the fields found in the input format, perhaps in a different order. Alternatively, the output format may contain input fields that have been changed. For example, a suffix or prefix added or a mathematical calculation applied.

The WMQI MRM formatter provides a wealth of options to specify how an output format may be built from an input format. In the many cases, the output format options are sufficient to produce the desired results. For the instances where an output format cannot be determined from the available set of reformatting options, a user exit may be used. A user exit is a segment of user written code (as opposed to IBM WMQI supplied code), which is invoked during the processing of a reformat. The user exit can perform the reformatting of the input message on behalf of WMQI. This powerful technique provides limitless options for generating appropriate output formats. The user exit has access to all the fields determined in the original input formatting and also has the ability to make operating system or other application calls such as accessing rows in a database, making MQSeries MQI calls and more.

Overview and usage of the bulk load utility

A certain level of knowledge of the MRM is required to make effective use of the bulk load utility. The bulk load utility will perform some checking of the input but it is certainly possible to create message set entries that are invalid and could not be created with the control center GUI. In most cases, these problems will result in deployment errors.

One way to find invalid or conflicting entries is to check out the elements and types in a message set and then attempt to check them back in.

Chapter 2. Installation

To install the utilities, merely unzip the distribution file and move the executable programs to an executable directory. When the distribution file is unzipped, the following files should be produced:

id06.pdf	(This documentation file)
mqsibulkimp2.exe	(bulk import utility executable)
mqsilistmsgset.exe	(message set listing utility executable)
mqsidelmsgset2.exe	(message set delete utility executable)
mt100def.xml	(sample input file)
acorddef.xml	(sample input file)
bulktst2.xml	(sample input file)

Installation verification

Three sample input files are provided with this utility. The sample files are as follows:

- mt100def.xml
- acorddef.xml
- bulktst2.xml

The first sample file illustrates is a tagged delimited example with two tab definitions. The second file is a tagged fixed length example. The third example is strictly artificial and includes three tab definitions.

To use any of the examples, use the following instructions.

1. Create a message set using the Control Center GUI. Check the message set in.
2. Stop the configuration manager (mqsistop configmgr).
3. Run the bulk load import utility using the desired sample input file.
4. Start the configuration manager (mqsistart configmgr).
5. Return to the Control Center GUI. To see the tabs that were created on the message set display, check out the message set and then check it in. The tabs should now appear in the Control Center display.
6. To view the elements, messages, values and/or types that were added, select the desired type of item and click the right mouse button. Select add to workspace and then select the desired items (ctrl-A will select all the items). Press finish. Expand the item displays as desired.

Chapter 3. Using the Bulk Load Utility

In order to use the bulk load utility, a message set must first be created using the WMQI control center. An input file must be created containing the elements, types, values, messages and wire format tabs that are to be added to the message set. Once the message set and definitions file have been created, then the import utility can be executed by using the command line syntax below.

Command Format

```
mqsimrmbulkimp –i –n <database> -u <user-id> -p <password> -s <message set> -f <file>
```

The database name, user-id, password and message set parameters can also be specified on the MRMImport element in the parameters file. If they are specified in both places, the command line specification will be used.

For example, to import the definitions contained in a specification file called defs1.txt into a message set called Test, with an MRM database name of MQSIMRDB, and database user id and password of db2admin, the following command line could be used:

```
mqsimrmbulkimp –I –n MQSIMRDB –u db2admin –p db2admin –s Test –f defs.txt
```

If wire format tabs are created during the import, the message set must be checked out and then checked back in before they will be visible on the message set display.

All output is written to the console. It is a good idea to redirect the output to a file so that it can be reviewed later.

Definition of message set components

The various components of an MRM message are defined using an XML based input file. The following XML tag names are the only ones that are allowed in the input file:

Element
Type
Message
Value
MRMImport
TDSTab
CWFTab
XMLTtab

Each XML tag uses a set of attributes to specify the various characteristics of the item being defined. All tag and attribute names, and most attribute values are case sensitive.

The element, type, message and value tags define the corresponding resources in the MRM. Each entity that is defined must have a unique identifier. The MRM requires that identifiers be unique within the entire message set. No two entities within a message set can have the same identifier, including entities of different types, such as an element and a type.

MRM identifiers must start with a letter or an underscore and may only contain letters (upper and lower-case), numbers and the underscore character. Element identifiers are used to refer to a particular element within ESQL statements. MRM names can be any combination of printable characters and are only used within the graphical user interface, and there are no restrictions on duplicate names.

Attributes for the Element tag

The element tag can have the following XML attributes specified. The attributes specify various characteristics of the element.

ident

The ident attribute specifies the MRM identifier to be assigned to this element. MRM identifiers must be unique within a message set. The identifier is used to refer to the element within esql statements, and is used as a default value for XML tag names. All element definitions must contain either an ident attribute or a ref attribute. The ident and ref attributes are mutually exclusive.

name

The name attribute specifies the MRM name to be assigned to this element. MRM names are the values that are displayed in most of the MRM GUI panels. The name attribute can only be specified with the ident attribute. It cannot be used with the ref attribute.

Ref

If an element is to be used in more than one place in a message definition, it must be defined as a global element and referenced in all other locations. The location where the element is defined should include an ident attribute to provide a name for the element. This name is then referenced by specifying elements in other locations that include a ref attribute rather than an ident attribute.

The ref attribute cannot be used with the ident or name attribute. In general, the characteristics of an element must be set where the element is defined, and cannot be specified where the element is referenced. The maxOccurs and minOccurs attributes are exceptions to this rule.

type

The type attribute must be either a simple type or a compound type reference. Compound types must be defined within the message set. The simple types are built-in and consist of seven types, namely string, integer, float, Boolean, binary, datetime and decimal. Simple types can be specified using the following names:

String	string	str	50105
Integer	integer	int	50106
Float	float	flt	50107
Binary	binary	bin	50109
Boolean	boolean	bool	50213
Datetime	datetime	date	50222
Decimal	decimal	dec	50224

physicalType

The physical type attribute is used to define the output and input format of an element when that element is used within a particular type. The element itself must be of a simple type. The physical type is not required for simple types of binary or Boolean, since these must have physical types of binary or Boolean respectively. Therefore, physical type cannot be specified if the element type is binary or Boolean. Physical type must be one of the following:

- Fixed Length, fixed, fixlen or fix
- Length Encoded String 1, Encoded 1 or enc1
- Length Encoded String 2, Encoded 2 or enc2
- `Null Terminated, Null Term or null`
- `Binary or bin`
- `Boolean or bool`

- `Integer or int`
- `Float`
- `Time Seconds, Seconds or sec`
- `Time Milliseconds, Milliseconds or mil`
- `Packed Decimal, Packed or pack`
- Extended Decimal, Decimal or dec

maxOccurs

The maxOccurs attribute specifies the maximum number of occurrences of a particular element that can occur at a particular location within a given type. The maxOccurs attribute can only be specified within a type definition. It cannot be specified in a global element definition.

minOccurs

The minOccurs attribute specifies the minimum number of occurrences of a particular element that can occur at a particular location within a given type. The minOccurs attribute can only be specified within a type definition. It cannot be specified in a global element definition.

repeat

The repeat attribute specified either a fixed number of occurrences or the name of an integer element that represents a variable number of occurrences. If the first character of the repeat attribute value is a number, then the repeat attribute is assumed to be a count. Otherwise, it is assumed to be the name of an integer element. If a count is specified, it must be a positive integer.

tag

The tag attribute is a string value that is used to identify an element or structure within a tagged message (either tagged delimited or tagged fixed length). Tags can be of a fixed length or can be delimited. If tags are defined as fixed length, then the value specified for the tag attribute must match the tag length specified in the type.

length

The length attribute specifies the length of a fixed length element. Length can only be used with elements of a simple type, except for Boolean simple types that cannot have a length specified. If specified, the length must be a positive integer greater than zero.

lengthRef

The length reference attribute is used with tagged delimited and custom wire formats. The value of this entry must be the identifier of an integer field that precedes the field in the same type.

lengthUnits

The length units attribute is primarily for use with DBCS and multi-byte character sets. It also allows the length of an element to be set to be the remaining number of characters in the message. If the end of bitstream option is selected, the element must be the last element in the message. If specified, the length units property must be one of the following:

- Bytes or 0
- Characters, char or 1
- Character Units, unit or 2
- End of Bitstream, end or 3

The default value is Bytes. If the End of Bitstream option is selected, the element should be the last element in the message.

justification

The justification attribute can be set for string and numeric values, to override the normal defaults of left for strings and right for numbers. The value for justification can be set to “L”, “Left” or “Left Justify” for left justification, “R”, “Right” or “Right Justify” for right and “N”, “None” or “Not Applicable” for no justification specified. This attribute applies to TDS and CWF renderings. For CWF formats, this property is only valid for fixed length strings and for numbers that are rendered as extended decimal.

padChar

The pad character attribute is used to specify the character that is to be used to pad fixed-length string and numeric values. The padding will be on the left or right depending on the justification specified. The pad character must be specified as one of the following:

- SPACE
- ZERO
- Character enclosed in single quotes

interpretValue

The interpret value is used for certain specialized field types that are defined in certain industry standards. The following values are the only values that will be accepted:

None
EDIFACT Service String
X12 Service String
Message Key

If present, the attribute value must match one of the above exactly, including upper and lower case characters.

repeatElemDelim

The repeatElemDelim attribute sets the character string that separates individual values for a particular element. This value can be any valid string. Carriage returns should be set using the four characters <CR> and line feeds should be set using the four characters <LF>.

decimalPoint

The decimal point attribute must be an integer, and is only valid for elements that are simple numeric types (decimal and float). The decimal point represents the position of an implied decimal point. If the value is set to a negative number, then extra zeros are imputed to the end of the number (e.g. if decimal point is set to –4, then a decimal value of 123 would be interpreted as 1230000). If this value is zero (default), then the precision property is checked for the formatting of the number.

The decimal point attribute is valid used for both tagged delimited and custom wire format inputs and outputs.

precision

If specified, the precision attribute must be a non-negative integer, and is only valid for elements that are numeric simple types of decimal or floating point. The precision property specifies then number of significant digits that will be included when a numeric value (decimal or floating point) is rendered. For example, if precision is set to 3, the value 123.4567 would be truncated to 123.456, while the value 123.4 would be rendered as 123.400. This property is only used with tagged delimited wire formats.

byteAlignment

The byte alignment property indicates the alignment of a character string. This property is only used with custom wire formats. The byte alignment must be specified as a number. It must be 1, 2, 4, 8 or 16. The default is 1.

format

The format attribute is used to determine the appearance of date and time values. It is only valid for elements that are of type datetime. This attribute is only used for custom wire formats.

formatString

The format attribute is used to determine the appearance of date and time values. It is only valid for elements that are of type datetime. This attribute is only used for tagged delimited and XML wire formats.

signed

The signed attribute applies to CWF renderings and indicates if there is a sign character. If specified, the value of this attribute must match one of the following:

None
Included Leading, Incl Lead or 1
Included Trailing, Incl Trail or 2
Separate Leading, Sep Lead or 3
Separate Trailing, Sep Trail or 4

The default for this attribute is None.

signOrientation

The sign orientation can be leading or trailing, and is only valid for numeric data items (decimal and floating point). Sign orientation is only used for tagged delimited wire formats. If specified, it must match one of the following values:

Leading
Trailing
None

positiveSign

This field indicates the string that should be added to a numeric value to indicate a positive number. It can be any valid character or string of characters. This property is only used for tagged delimited wire formats. The sign orientation should be set to either Leading or Trailing if this property is set.

negativeSign

This field indicates the string that should be added to a numeric value to indicate a negative number. It can be any valid character or string of characters. This property is only used for tagged delimited wire formats. The sign orientation should be set to either Leading or Trailing if this property is set.

TDSnullKey

This field is used to specify the output format of an element with a null value. If specified, the value of this attribute must be one of the following:

- NULLPadFill, pad or 0
- NULLLogicalValue, logical or 1
- NULLLiteralValue, literal or 2

This field may not be specified for elements of type Boolean or binary. It applies to any tagged delimited tabs that are defined within the message set.

TDSnullValue

This field is used to specify the value that is to be used when the TDSnullKey value is set to NULLLogicalValue or NULLLiteralValue. This field must be a string and may need to be the same length as the element. This attribute may not be specified for elements of type Boolean or binary. It applies to any tagged delimited tabs that are defined within the message set.

CWFnullKey

This field is used to specify the output format of an element with a null value. If specified, the value of this attribute must be one of the following:

- NULLPadFill, pad or 0
- NULLLogicalValue, logical or 1
- NULLLiteralValue, literal or 2

This field may not be specified for elements of type Boolean or binary. It applies to any custom wire format tabs that are defined within the message set.

CWFnullValue

This field is used to specify the value that is to be used when the CWFnullKey value is set to NULLLogicalValue or NULLLiteralValue. This field must be a string and may need to be the same length as the element. This attribute may not be specified for elements of type Boolean or binary. It applies to any custom wire format tabs that are defined within the message set.

xmlName

This field indicates the string value that should be used for the tag name when this element is rendered as XML output. The XML name can be specified on the element definition, and can be overridden in any location where the element is referenced. The XML name should be a valid XML tag name. It can include a colon for qualified names.

xmlFormat

This field indicates the string value that should be used to specify the format of a date field when this element is rendered as XML output. The XML format can be specified on the element definition, and can be overridden in any location where the element is referenced.

xmlRender

This field indicates how a particular element should be rendered when the element is rendered as XML output. The element can be rendered as an XML element or attribute. There are also options to specify a rendering where the element name is actually rendered as an attribute value. The XML rendering can be specified on the MRM element definition, and can be overridden in any location where the MRM element is referenced.

If specified, the XMLrender attribute should be specified as one of the following:

- element, elem or 0
- attribute, attr or 1
- elementAttrId, attrId or 2
- elementAttrVal, attrVal or 3
- elementAttrIdVal, attrIdVal, idVal or 4

xmlIDattrName

This field is used with XML rendering types of elementAttrId (“2”) and elementAttrIdVal (“4”) only. It sets the Member ID Attribute Name property. It is a character string that contains a valid attribute name. If not specified, this field defaults to “ID”.

xmlIDattrValue

This field is used with XML rendering types of elementAttrId (“2”) and elementAttrIdVal (“4”) only. It sets the Member ID Attribute Value property. It is a character string that contains a valid attribute name. If not specified, this property will default to the MRM identifier of the child element.

xmlValAttrName

This field is used with XML rendering types of elementAttrId (“3”) and elementAttrIdVal (“4”) only. It sets the Member Value Attribute Name property. It is a character string that contains a valid attribute name. If not specified, this field defaults to “val”.

Attributes for the Type tag

ident

The ident attribute specifies the MRM identifier to be assigned to this type. MRM identifiers must be unique within a message set. The identifier is used to refer to the type within the MRM. All type definitions must contain either an ident attribute or a ref attribute. The ident and ref attributes are mutually exclusive.

name

The name attribute specifies the MRM name to be assigned to this type. MRM names are the values that are displayed in most of the MRM GUI panels. The name attribute can only be specified with the ident attribute. It cannot be used with the ref attribute.

ref

If an element is to be used in more than one place in a message definition, it must be defined as a global element and referenced in all other locations. The location where the element is defined should include an ident attribute to provide a name for the element. This name is then referenced by specifying elements in other locations that include a ref attribute rather than an ident attribute.

The ref attribute cannot be used with the ident or name attribute. In general, the characteristics of a type must be set where the type is defined, and cannot be specified where the type is referenced. The maxOccurs and minOccurs attributes are exceptions to this rule.

baseType

The baseType attribute is used where an element has both a simple value and child elements. The base type defines the data type of the value. The base type must be a simple MRM type. See the type attribute under the Element tag for valid simple types.

composition

This attribute sets the MRM composition field. Valid values for composition include empty for empty, choice for choice, simple, simple_unord or simple_unordered for simple unordered, unord, unorder or unordered for unordered, ord, order or ordered for ordered, seq or sequence for sequence, and msg or message for message. The composition value can be in any case.

dataElemSep

The dataElemSep attribute sets the data element separation value for a type. This value can be any valid string. Carriage returns should be set using the four characters <CR> and line feeds should be set using the four characters <LF>.

delimiter

The delimiter attribute sets the delimiter value for a tagged type. This value can be any valid string. Carriage returns should be set using the four characters <CR> and line feeds should be set using the four characters <LF>.

groupInd

The groupInd attribute sets the delimiter value for a tagged type. This value can be any valid string. Carriage returns should be set using the four characters <CR> and line feeds should be set using the four characters <LF>.

groupTerm

The groupTerm attribute sets the delimiter value for a tagged type. This value can be any valid string. Carriage returns should be set using the four characters <CR> and line feeds should be set using the four characters <LF>.

tagDataSep

The tagDataSep attribute sets the character string that separates the data portion of a tag from the data for a tagged delimited type. This value can be any valid string. Carriage returns should be set using the four characters <CR> and line feeds should be set using the four characters <LF>.

tagLen

The tagLen attribute sets the length of a fixed length tag. Tags can be either fixed length or must have a delimiter at the end of the tag. This value is only valid for tagged types. If specified, this value must be a positive integer.

typeContent

The typeContent attribute sets the type content value for a type. Type content can be open, open defined (specify as defined) or closed. The value can be specified in any case (upper, lower or mixed). The default type content is closed.

Attributes for the Value tag

ident

The ident attribute specifies the MRM identifier to be assigned to this value. MRM identifiers must be unique within a message set. The identifier is used to refer to the value within the MRM. All value definitions must contain either an ident attribute. Values can be defined where they are used or can be referenced from other locations within the definition file. The ident and ref attributes are mutually exclusive.

name

The name attribute specifies the MRM name to be assigned to this value. MRM names are displayed in most of the MRM GUI panels. The name attribute can only be specified with the ident attribute. It cannot be used with the ref attribute.

ref

If a value is to be used in more than one place in a message definition, it must be defined in one location and referenced in all other locations. The location where the value is defined should include an ident attribute to provide a name for the element. This identifier is then referenced by specifying values in other locations that include a ref attribute rather than an ident attribute.

The ref attribute cannot be used with the ident or name attribute. The characteristics of a value must be set where the value is defined, and cannot be specified where the value is referenced. The role attribute is an exception to this rule.

role

The role attribute specifies the role that this value is being used in when assigned to a specific element.

type

The type attribute indicates the simple type that the value represents. The types allowed are the same as the type attribute for an element.

value

The value attribute specifies the value to be assigned to this value item.

Attributes for the Message tag

ident

The ident attribute specifies the MRM identifier to be assigned to this message. MRM identifiers must be unique within a message set. The identifier is used to refer to the message within the MRM and on the MessageType parameter for input and output messages. All message definitions must contain either an ident attribute or a ref attribute. The ident and ref attributes are mutually exclusive.

name

The name attribute specifies the MRM name to be assigned to this message. MRM names are displayed in most of the MRM GUI panels. The name attribute can only be specified with the ident attribute. It cannot be used with the ref attribute.

ref

If a message is to be used in more than one place in a message definition, it must be defined in one location and referenced in all other locations. The location where the message is defined should include an ident attribute to provide an identifier for the message. This identifier is then referenced by using messages in other locations that include a ref attribute rather than an ident attribute.

The ref attribute cannot be used with the ident or name attribute. In general, the characteristics of a message must be set where the message is defined, and cannot be specified where the message is referenced. The maxOccurs and minOccurs attributes are exceptions to this rule.

type

The type attribute should specify the name of a compound type that is the highest-level type within the message.

messageKey

The message key attribute specifies the message key attribute for a message entry in the MRM repository. The message key can be any valid string. This parameter is ignored in the current release.

Attributes for the MRMImport tag

msgPrefix

The message prefix attribute specifies a character string that is inserted in front of each message identifier. For example, if the msgPrefix attribute is set to “m_”, then a message whose ident attribute is set to Orders would result in an MRM identifier of “m_Orders”.

msgSuffix

The message suffix attribute specifies a character string that is added to the end of each message identifier. For example, if the msgSuffix attribute is set to “_msg”, then a message whose ident attribute is set to Orders would result in an MRM identifier of “Orders_msg”.

elemPrefix

The element prefix attribute specifies a character string that is inserted in front of each element identifier. For example, if the elemPrefix attribute is set to “e_”, then an element whose ident attribute is set to OrderNumber would result in an MRM identifier of “e_OrderNumber”.

elemSuffix

The element suffix attribute specifies a character string that is added to the end of each element identifier. For example, if the elemSuffix attribute is set to “_elem”, then an element whose ident attribute is set to OrderNumber would result in an MRM identifier of “OrderNumber_elem”.

typePrefix

The type prefix attribute specifies a character string that is inserted in front of each type identifier. For example, if the typePrefix attribute is set to “t_”, then a type whose ident attribute is set to OrderAddress would result in an MRM identifier of “t_OrderAddress”.

typeSuffix

The type suffix attribute specifies a character string that is added to the end of each type identifier. For example, if the typeSuffix attribute is set to "_type", then a type whose ident attribute is set to OrderAddress would result in an MRM identifier of "OrderAddress_type".

valuePrefix

The value prefix attribute specifies a character string that is inserted in front of each value identifier. For example, if the valuePrefix attribute is set to "v_", then a value whose ident attribute is set to len10 would result in an MRM identifier of "v_len10".

valueSuffix

The value suffix attribute specifies a character string that is added to the end of each value identifier. For example, if the valuePrefix attribute is set to "_val", then a value whose ident attribute is set to len10 would result in an MRM identifier of "len10_val".

MRM_DB

This attribute should contain the name of the MRM repository database. This value can be specified on the command line with the –n parameter. If no database name is specified on the command line, then this attribute is used for the name of the MRM repository database. This attribute should contain a string that is less than 64 characters in length and is a valid database name.

DBUser

This attribute should contain the user id that will be used to access the MRM repository database. If specified, the user id must have update authority on the MRM repository database. This parameter is required if the user id that the bulk import utility is running under does not have the necessary database permissions. The value specified in this parameter can be overridden with a command line parameter (-u).

DBPw

This attribute should contain the password that will be used to access the MRM repository database. If specified, the password must match the user id specified in the DBUser attribute. This parameter is required if the user id that the bulk import utility is running under does not have the necessary database permissions and an alternative user id and password are required. The value specified in this parameter can be overridden with a command line parameter (-p).

schema

This attribute should contain the user id that will be used to access the MRM repository database. If specified, the user id must have update authority on the MRM repository database. This parameter is required if the user id that the bulk import utility is running under does not have the necessary database permissions. The value specified in this parameter can be overridden with a command line parameter (-u).

messageSet

This attribute should contain the name of the message set that the definitions are to be imported into. The value specified in this parameter can be overridden with a command line parameter (-s).

Attributes for the CWFTab tag

For each CWFTab tag that is found, a custom wire format tab will be added to the message set, unless a CWF tab with the same name already exists. If a CWF tab with the same name already exists, a warning message will be issued.

name

The name attribute specifies the MRM name to be assigned to this tab. This name will be displayed on the tab associated with this output format on the GUI panels. This attribute is required.

ident

The ident attribute specifies the MRM wire format identifier to be assigned to this output format. The wire format identifier must be unique within a message set. The identifier is used to identify the wire format to be used for input or output parsing operations. This parameter can be specified in the default parameters for an MQInput node or in an RFH2 header. For output operations, the identifier is specified in the Properties folder.

Attributes for the TDSTab tag

For each TDSTab tag that is found, a custom wire format tab will be added to the message set, unless a TDS tab with the same name already exists. If a TDS tab with the same name already exists, a warning message will be issued.

name

The name attribute specifies the MRM name to be assigned to this tab. This name will be displayed on the tab associated with this output format on the GUI panels. This attribute is required.

ident

The ident attribute specifies the MRM wire format identifier to be assigned to this output format. The wire format identifier must be unique within a message set. The identifier is used to identify the wire format to be used for input or output parsing operations. This parameter can be specified in the default parameters for an MQInput node or in an RFH2 header. For output operations, the identifier is specified in the Properties folder.

msgStd

This attribute sets the messaging standard for the TDS tab. The messaging standard affects a number of default settings for other parameters. Valid values for the messaging standard are UNKNOWN, SWIFT, ACORD AL3, EDIFACT and X12. The default is UNKNOWN.

groupInd

This attribute sets the default group indicator parameter for this tab. The group indicator applies to compound types and can be any string value of one or more characters. The carriage return and line feed parameters are specified as <CR> and <LF> respectively.

groupTerm

This attribute sets the default group terminator parameter for this tab. The group terminator applies to compound types and can be any string value of one or more characters. The carriage return and line feed parameters are specified as <CR> and <LF> respectively.

tagDataSep

This attribute sets the default tag data separator parameter for this tab. The tag data separator is the character that separates a tag from subsequent data. The tag data separator applies to compound types and can be any string value of one or more characters. Tags must either have a delimiter sequence or must be of fixed length. The carriage return and line feed parameters are specified as <CR> and <LF> respectively.

tagLen

This attribute sets the default tag length parameter for this tab. Tags must either have a delimiter sequence or must be of fixed length. The tag length parameter must be specified as a positive integer.

decimalPoint

This sets the decimal point character to be used for this wire format. The decimal point attribute should consist of a single character that is to be used to separate the whole part of a number from the fractional part, for numeric types that contain non-integer values.

delimiter

The delimiter represents a character sequence that indicates the end of one data item and the beginning of the next data item. The delimiter can be any string value of one or more characters. The carriage return and line feed parameters are specified as <CR> and <LF> respectively.

trimFix

The trimFix attribute indicates whether leading and/or trailing blanks and white space characters should be trimmed from fixed length string elements. The trimFix attribute can be set to *No Trim*, *Leading White Spaces*, *Leading* or *Lead* for *Leading White Space*, *Trailing White Spaces*, *Trailing* or *Trail* for *Trailing White Spaces*, *Trim Both* or *Both* for *Trim Both*, and *Trim Padding Chars* or *Pad* for *Trim Padding Chars*.

escapeChar

The escape character specifies the character that allows other special characters to appear in the data portion of an item. The escape character is specified as a single ASCII character. The carriage return and line feed parameters are specified as <CR> and <LF> respectively. The escape character attribute cannot be specified if the reserved characters attribute is set???????

reservedChars

The reserved characters property specifies a list of characters that must be preceded by an escape character when used in the data part of a data item. The values of the reserved items can include special character sequences contained within angled brackets that can be filled in dynamically at run time.

outCompress

The output compression property can specify a value of *SimpleAL3CharCompression* or *AL3* for *SimpleAL3CharCompression*. No other values can be specified.

inCompress

The input compression property can specify a value of *SimpleAL3CharCompression* or *AL3* for *SimpleAL3CharCompression*. No other values can be specified.

boolTrue

This property specifies a character string of one or more characters that is used in input and output data to represent a Boolean true value.

boolFalse

This property specifies a character string of one or more characters that is used in input and output data to represent a Boolean false value.

boolNull

This property specifies a character string of one or more characters that is used in input and output data to represent a Boolean null value. A null value indicates something that is neither true nor false. For example, it can represent something that cannot be determined or is unknown.

Attributes for the XMLTab tag

For each XMLTab tag that is found, a custom wire format tab will be added to the message set, unless a XML tab with the same name already exists. If a XML tab with the same name already exists, a warning message will be issued.

name

The name attribute specifies the MRM name to be assigned to this tab. This name will be displayed on the tab associated with this output format on the GUI panels. This attribute is required.

ident

The ident attribute specifies the MRM wire format identifier to be assigned to this output format. The wire format identifier must be unique within a message set. The identifier is used to identify the wire format to be used for input or output parsing operations. This parameter can be specified in the default parameters for an MQInput node or in an RFH2 header. For output operations, the identifier is specified in the Properties folder.

supXmlDecl

The suppress XML declaration property determines whether an XML declaration is generated for output XML documents. If specified, the value of this attribute must be a character string of one or more characters starting with either the letter “Y” or the letter “N”, in either upper or lower-case. Any character string starting with a “Y” is interpreted as “Yes”, while an “N” is interpreted as “No”. The default value within the MRM repository for this parameter is “No”.

standaloneDoc

The standalone document property indicates whether or not to include the standalone attribute within an XML declaration. If specified, the standaloneDoc attribute must be a character string of one or more characters starting with either the letter “Y” or the letter “N”, in either upper or lower-case. Any character string starting with a “Y” is interpreted as “Yes”, while an “N” is interpreted as “No”. The default value for this parameter is “No”. If no standalone attribute is desired within the XML declaration, then this attribute should be omitted.

supDocType

The suppress doctype property indicates whether a doctype specification should be included in the output instance document that is generated. If specified, the supDocType attribute must be a character string of one or more characters starting with either the letter “Y” or the letter “N”, in either upper or lower-case. Any character string starting with a “Y” is interpreted as “Yes”, while an “N” is interpreted as “No”. The default value for this parameter is “No”.

doctypeSystemId

The doctype system id property is a string value that will be used as the value for a system id attribute in a doctype specification that is to be included in the output instance document that is generated. If the supDocType attribute is set to “Yes”, then this parameter is ignored by the MRM and cannot be altered in the control center user interface.

doctypePublicId

The doctype public id property is a string value that will be used as the value for a public id attribute in a doctype specification that is to be included in the output instance document that is generated. If the supDocType attribute is set to “Yes”, then this parameter is ignored by the MRM and cannot be altered in the control center user interface.

doctypeText

The doctype text property specifies additional text to be included in the doctype specification. The text has a maximum length of 32K characters. New line characters must be replaced with a “\n” escape sequence and tab characters must be replaced with a “\t” escape sequence respectively. The text specified in this attribute must be valid XML characters. Normal XML escape sequences must be used (e.g. &amp; for the ampersand character, etc).

rootTagName

The root tag name property specifies the name of the root tag for the XML instance document. It can be specified as any string containing a valid XML tag name. The name may be followed by a space and additional text for attribute/value pairs to appear with the root tag. Please remember that the quotation marks surrounding any such attribute/value pairs must use the normal XML escape characters, since they are considered data characters within the parameters file. The escape sequences will be replaced in the data that is placed in this field within the MRM repository. The default value for this parameter is “*MRM*”.

enableVersion

This attribute 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. If specified, the value of this attribute must be a character string of one or more characters starting with either the letter “Y” or the letter “N”, in either upper or lower-case. Any character string starting with a “Y” is interpreted as “Yes”, while an “N” is interpreted as “No”. The default value within the MRM repository for this parameter is “Yes”.

boolTrue

This property specifies a character string of one or more characters that is used in input and output data to represent a Boolean true value.

boolFalse

This property specifies a character string of one or more characters that is used in input and output data to represent a Boolean false value.

encodeNullNum

This property specifies the encoding rules for 'null' values in numeric elements. It must contain one of the following values:

- NULLAttribute
- NULLEmpty
- NULLValue
- NULLElement
- NULLValAttr

encodeNullNumVal

When the Encoding Null Num parameter is set to NULLAttribute, NULLValue or NULLElement, this attribute specifies the value of the Encoding Null Num parameter, otherwise it is ignored and cannot be changed in the control center (the field is grayed-out).

encodeNullNonNum

This property specifies the encoding rules for 'null' values in non-numeric elements. It must contain one of the following values:

- NULLAttribute
- NULLEmpty
- NULLValue
- NULLElement
- NULLValAttr

encodeNullNonNumVal

When the Encoding Null Non-Num parameter is set to NULLAttribute, NULLValue or NULLElement, this attribute specifies the value of the Encoding Null Non-Num parameter, otherwise it is ignored and cannot be changed in the control center (the field is grayed-out).

Chapter 4. Usage Considerations

Checking of values in MRM fields

Although a considerable amount of checking is performed on values that are imported into a message set, it is not possible to reject all entries that result in an invalid entry in one or more MRM items. Any such errors usually show up when an attempt is made to work with the message set. This can include deploy operations, checking items in and out and attempts to export the message set.

In many cases, checking out the message set and then attempting to freeze it can locate invalid entries in the MRM. This is in fact what deployment and export operations do. If the freeze operation is successful, then the message set should be unfrozen and checked back in.

Errors can also appear when an item is checked out and then checked back in. Items are validated when a check in operation is performed.

XML verification of input parameters file

The bulk load utility expects to process well-formed XML documents. The input definitions file must be a well-formed XML document.

The utility will attempt to reject XML documents that are invalid or are mal-formed in a graceful manner, producing one or more appropriate error messages. However, if a file does not produce a meaningful error message or ends with an error, please check that the input file is in fact well formed, load the document into Microsoft Internet Explorer V5.0 or greater or some other XML tool.

The author is interested in cases where the bulk load utility terminates abnormally, but cannot guarantee that a solution will be provided.

Adding entries to a Message Set that is not empty

The bulk load utility is designed to work with WMQI message sets that have no previously defined elements, compound types, values and messages. It is designed to work with previously defined wire format tabs, but these can also be added using the utility. The utility cannot be used to update or change the definitions of existing items.

In the current version, any conflicts with existing items will result in an error, as any references to any existing item will not be resolved.

If an item with the same identifier already exists within the message set, and the item appears to be the same as the one that is being defined, then the utility will use the existing item rather than defining a new one. It must be emphasized that the checking that the two items are the same is very limited. For elements, a check is made that the types are the same. If the existing item is defined as having a simple type, then the types of the new and existing elements must be the same. If the existing item is defined as a compound type, then the identifiers of the compound types of the new and existing types must be the same.

No attempt is made to update an existing item with the same identifier. If the existing item is clearly different, then an error will be generated and the entire load will fail. Otherwise, a warning message is produced and the load will use the existing item. All attributes of an existing element that is referenced will be left as previously defined.

If a wire format tab is added to a message set, no changes will be made to existing elements. This may result in elements that do not have valid default values for the new tab. For example, if a CWF tab is added, and an element is already a member of a compound type, then the element will not have a valid length for the new tab and any attempt to check in or deploy the message set will now fail until the length is added manually with the GUI.

Wire formats and element attributes

The MRM in WMQI V2.1 supports three different wire format protocols, known as custom wire format (CWF), tagged delimited (TDS) and XML. Messages can be rendered using one or more wire formats. The characteristics of the rendering are controlled by attributes of the element.

When an element is defined, the only basic attribute that the element contains is the type of data (e.g. integer, string, decimal, etc). All other characteristics (attributes) of an element are defined for the use of the element within a particular wire format. These attributes can either apply to the element itself or to the use of the element in a particular location within a compound type (known as a member attribute). The different wire formats are not consistent in their handling of element attributes. For example, a length value is defined as an attribute for tagged delimited wire formats, as a member attribute for custom wire formats and ignored for XML wire formats. This can lead to some confusion. It is recommended that all messages within a single message set use the same wire format, unless this cannot be avoided.

Unlike most tagged delimited and XML attributes, custom wire format element attributes are related to the use of an element in a type, rather than properties of the element itself. Therefore, it is possible to use the same element in more than one type but with different attributes. For example, in one place, an element could be specified as 10 characters long with left justification and padded with spaces, while the same element could be used in a different location and be specified as 20 characters long with right justification and padded with zeros.

Element attributes for tagged delimited and XML wire formats can generally be specified as properties of the element (attributes), and can be overridden when the element is used in a particular location within a type (member attributes).

The XML name, rendering and format attributes can be specified as attributes of an element, and can also be overridden when an element is used in a particular location.

Simple types and attributes of base type within a compound type

The MRM allows the specification of attributes for a simple type that is contained within a compound type and for a compound type with a base simple type that is contained within a compound type. The current release of the bulk import utility does not support this function. It is anticipated that this capability will be added in the next release of this SupportPac.

Embedded messages

Embedded messages are not supported in this release. It is anticipated that this capability will be added in the next release of this SupportPac.

Chapter 5. Additional utilities (List and Delete)

Two additional command line utilities are provided with this SupportPac. The list utility will list all message sets and messages that are defined within the MRM repository. The delete utility will delete a particular message set from the MRM repository. The delete utility is provided for cases where an invalid entry is made in the MRM repository, and the message set cannot be deleted from the control center GUI. The deletion function of the control center should be used whenever possible.

List utility syntax and usage

The list utility will list all message sets and messages that are defined within the MRM repository. It will list the name, identifier, next element id and project id for each message set defined within the repository. The message identifier and element id for each message within a message set will be listed under the message set that they are defined in.

The syntax of the message set list utility is as follows:

```
mqsilistmsgset -n MRM_Database_Name -u DB_Userid -p DB_Password {-h Schema_name}
```

The database name is the only required parameter. If the command is being run under a user id that does not have authorization to read the MRM repository database, then the user id and password parameters must be specified. If the database schema names are different than the user id that is specified, then the schema name of the tables within the MRM repository must also be provided. For example, if the MRM database is named MQSIMRDB, and it was defined with a user id of mqsiadmin and the password for mqsiadmin is pw, the following command could be used:

```
mqsilistmsgset –n MQSIMRDB –u mqsiadmin –p pw
```

Delete utility syntax and usage

The delete utility will delete all entries for a particular message set within the MRM repository database. It deletes the entries directly from the MRM repository database. It does not do any validation of any entries prior to deleting the message set, so it can delete entries that are corrupted or invalid.

The delete utility will accept either a message set name or an internal project identifier. Every message set that is defined in an MRM repository is assigned a unique project identifier. The project identifier is a number. The lowest project number is 100. The first 12 project identifiers are used for the repository data for the standard IBM MQSeries headers. User defined projects usually start with 112 and increase. Project numbers are never reused within an MRM repository. The list utility provided with this SupportPac will list the message set project numbers for all defined message sets within the repository. It is rarely necessary to use the project number rather than the message set name to identify a message set for deletion.

The syntax of the message set delete utility is as follows:

```
mqsidelmsgset2 –n DB_Name –u DB_user –p DB_password –s Message_Set {-h Schema}
```

The database name and message set name or identifier are the only required parameters. If the command is being run under a user id that does not have authorization to read the MRM repository database, then the user id and password parameters must be specified. If the database schema names are different than the user id that is specified, then the schema name of the tables within the MRM repository must also be provided. For example, to delete a message set named testmsgset, and assuming the MRM database is named MQSIMRDB, and the database was defined with a user id of mqsiadmin and the password for mqsiadmin is pw, the following command could be used:

```
mqsidelmsgset2 –n MQSIMRDB –u mqsiadmin –p pw –s testmsgset
```

When a message set is deleted from the repository, there may still be references to the message set in one or more control centers. These references should be removed (use the right mouse button and select the only option that is presented, namely remove).

Chapter 6. Problem Determination

Reporting problems

The author is interested in any reports of problems encountered when using this SupportPac. The author's email address is given on the front page of this document. To assist in problem diagnosis, the author will probably require the following:

- Description of the problem
- General description of the environment
- Copy of the import file and the utility command line parameters that lead to the problem
- Copy of the output that is produced by the utility – this usually requires the output to be redirected to a file. It is preferable to turn on tracing (-t –d and –v switches) when this output is produced.
- If the problem involves database issues, an exported copy of the repository database. To create this, create a temporary directory and issue the db2move command using the database name and the word EXPORT as the two parameters. Please zip the files that are created in the directory into a single file and include that with the problem report.

The author will attempt to provide a solution to problems that are reported.

Could not locate message set

The import utility could not locate the message set in the MRM repository. This usually means the message set has not been defined or that the MessageSet parameter in the parameters file is incorrect. Use the mqsilistmsgset utility (part of this SupportPac) to display the message sets that are defined in the MRM repository.

Problems with the repository database

If the data in the MRM repository database becomes corrupted, the problem is likely to show up as messages from the control center or the import/export utility program indicating an internal error in the message repository. Given the complexity of the MRM tables and the number of relationships between the data contained in the tables, these problems can be very difficult to diagnose.

The best approach to isolate the problem is to first try to check out the message set and freeze it. If the freeze is successful, then the message set should then be unfrozen and checked in. Individual elements and types can also be checked out and checked back in. If there are problems with the existing values that prevent an item from being checked in or updated, the item should be unlocked rather than checked in.

MRM repository background

A complete explanation of the inner workings of the MRM is beyond the scope of this SupportPac. This section will attempt to provide some information and techniques that might prove useful should problems arise with the repository database.

In the repository database, message sets are known as projects. Each message set is assigned a unique project identifier. Project identifiers are ascending numbers beginning with 100. The first 12 entries are usually taken by IBM supplied message definitions, so the first user-defined project identifier will usually be 112. Project numbers are assigned sequentially and are not reused. The next project identifier to be assigned is kept in the NEXT_PROJECT_KEY column in the REPOSITORY table. When a new message set is created, the message set will be assigned the value in this column as its unique project identifier and the value in the NEXT_PROJECT_KEY column will be increased by one.

The PROJECT table contains one row for each message set known to the repository. To display the project identifier and message set names for all message sets known to the message repository, start a DB2 command line processor session and enter the following statements:

```
CONNECT TO MQSIMRDB
SELECT PROJECT_KEY, NAME FROM DB2ADMIN.PROJECT
```

Please note that the repository database name is assumed to be MQSIMRDB and the database schema name is assumed to be DB2ADMIN. The schema name is usually the user id that the database was created with. If the schema name is not known, it can be displayed with the DB2 control center.

The project key is used in most of the other table entries in the repository database to tie the various rows to a particular message set. For example, elements, lengths and types must all contain a valid project value for the message set that they belong to.

Every element, length and type defined in a message set will be assigned an identifier. The identifiers are numbers and start at 100000 within each message set. The identifiers are assigned sequentially and are not reused. Each element, type and length will be assigned an identifier when it is created. The next identifier to be used for a particular message set is kept NEXT_ID column in the PROJECT table. The following SQL statement will display the project identifier, next identifier value and message set name for each message set defined in the message repository.

```
SELECT PROJECT_KEY, NEXT_ID, NAME FROM DB2ADMIN.PROJECT
```

The schema name (DB2ADMIN) may be different than this example and is not required if the command is being executed under the user id that the database tables were created under.

Every defined element must have a type entry. Elements can have either a simple type or a compound type. Five simple types are automatically defined within the message repository for all message sets. The types and their associated identifiers are as follows:

• String	50105
• Integer	50106
• Float	50107
• Binary	50109
• Boolean	50213

Integers are modeled as 32-bit integers and floating-point numbers are modeled as 64-bit entities.

Compound types will require an entry in the M_TYPE table. Each compound type entry will contain the project identifier that the type is a member of and an identifier assigned when it is created. Compound types must contain one or more elements. A compound type is considered incomplete if it does not contain any elements. The definition of which elements are associated with a particular compound type is made with rows in the TYPE_MEMBER table. Each row in the TYPE_MEMBER table will contain the project that the type is defined in, the type identifier (TYPE_ID), the identifier of the element (CHILD_ID) and a sequence number. The sequence number is used to keep the elements in the proper order within the type, and is an ascending number beginning with 1.

To display the type relationships for all compound types in a particular message set, first ascertain the project key as described above. Then issue the following SQL statement, following the same guidelines described above:

```
SELECT PROJECT, TYPE_ID, CHILD_ID, SEQUENCE_NUM FROM
DB2ADMIN.TYPE_MEMBER WHERE PROJECT=nnn
```

Attributes can be associated directly with an element, type or message, or they can be associated with the use of an item in a compound type (member attribute). Attributes in a compound type are associated with a type member identifier, to indicate a particular usage of an element, type or message within the type. Member attributes and attributes are associated with a particular wire format tab.

Diagnostic utilities

Two utilities are provided with this SupportPac to display a list of the message sets defined in the MRM repository and to delete a complete or partial message set.

These utilities should only be used if the message repository has become corrupt. They are not supported for normal use. They are provided solely for use in the event that a message set has become corrupted and cannot be deleted using the Control Center.

The first utility will list all message sets defined in the MRM repository, as well as any messages defined within each message set. The listing includes the internal project key and the next element identifier. This utility will only list message sets that have an entry in the PROJECT database. If the next element identifier is 100000, then there the message set is empty, with no defined elements, lengths, types and messages. The project key is the internal identifier by which the message set is identified in all other entries in the MRM message repository.

The list utility takes one to four parameters as input. It will write its output to standard out. The first parameter is the message repository database name (as defined to ODBC). This is the only required parameter. The second and third parameters are the user id and password to use when accessing the database. If a schema name is required, it should be specified as the fourth parameter. For example, if the MRM database name is MQSIMRDB, the user id is db2admin, the password is db2admin and the schema name is DB2ADMIN, the list command would look like the following:

```
mqsilistmsgset –n MQSIMRDB –u db2admin –p db2admin –h DB2ADMIN
```

The syntax of the delete command is similar. The first parameter is the name of the message set or the project key of the message set that will be deleted. If the message set entry in the PROJECT table exists, then the message set can be referenced by name. The name is case sensitive. If the PROJECT entry is missing or invalid, then the project key must be determined and any individual entries associated with this project key can be deleted by specifying the project key rather than the message set name. The next four parameters are the name of the database, the user id and password to use and the schema name if required. The last three parameters can be omitted if the command is being run from the user id that created the database. For example, to delete a message set named TestMsgSet in database MQSIMRDB using user id db2admin and password db2admin, with a schema name of DB2ADMIN, the command would look like the following:

```
mqsidelmsgset2 -s TestMsgSet –n MQSIMRDB –u db2admin –p db2admin –h DB2ADMIN
```

End of Document