# WebSphere MQ Integrator Tagged/Delimited message examples

# Version 1.0

December, 2001

Peter Edwards
pedwards@uk.ibm.com

Simon Gormley
sgormley@uk.ibm.com

IBM United Kingdom Laboratories
Hursley Park, Hursley
Winchester SO21 2JN
United Kingdom

**Take Note!**

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

**First Edition, December 2001**

This edition applies to Version 1.0 of *WebSphere MQ Integrator - Tagged/Delimited message examples* and to all subsequent releases and modifications unless otherwise indicated in new editions.

# Table of Contents

# Notices

The following paragraph does not apply in any country where such provisions are inconsistent with local law.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

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 each item has been reviewed by IBM 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 MQ
- WebSphere MQ Integrator

The following terms are trademarks of other companies:

- Windows NT Microsoft Corporation
- S.W.I.F.T. S.W.I.F.T. s.c.r.l., Avenue Adèle 1, 1310 La Hulpe, Belgium

# Summary of Amendments

| Date | Changes |
| --- | --- |
| 14 December 2001 | Initial release |

# Preface

This SupportPac aims to introduce the message designer to possible ways of using WebSphere MQ Integrator V2.1 to model Tagged/Delimited messages belonging to existing message standards such as SWIFT and other message standards, using the MRM message model.

# Bibliography


- *WebSphere MQ Integrator  Working With Messages*, IBM Corporation. SC34-6039
- *WebSphere MQ Integrator  Control Center Online Help*, IBM Corporation.

# Chapter 1. Introduction

## Overview

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 Tagged/Delimited format is one such message format supported by WMQI, which can use tags, delimiters, or both to define message structure. The Tagged/Delimited Wire Format layer in WMQI can be customized to support several different methods of formatting data. These methods are known as Data Element Separation techniques, and the methods supported by WMQI are: Fixed Length, Variable Length Elements Delimited, All Elements Delimited, Tagged Delimited and Tagged Fixed Length.

Fixed Length data element separation requires the length of each element in the message to be known. This information is then used to separate the message elements. The length can be defined either at design time, or by using data present in the message.

Variable Length Elements Delimited and All Elements Delimited methods both use a *Delimiter* to mark the end of an element. A Delimiter is a constant which separates the value of one element of data from the value of another element of data. When using All Elements Delimited, every element present in the message is separated from the next by a Delimiter. Variable Length Elements Delimited messages only use Delimiters to signify the end of elements of data whose lengths are not defined. i.e. if an element has a length defined there will be no delimiter to mark the end of the element, as this can be determined using the length.

Tagged Delimited and Tagged Fixed Length messages both use *Tags* to identify elements. A Tag is a string that precedes the data value of an element, and is unique to that element within a message set. The end of the tag is determined by one of two techniques: a Tag Data Separator, or using Tag Length. A Tag Data Separator is similar to the Delimiter described previously, except that it is used to signify the end of the tag, and start of the data. Similarly using Tag Length is similar to the Fixed Length data element separation technique, as the length of the tag is known, the start of the element data can be deduced. Using the Tag Length method to determine the end of the tag requires all of the tags be of the same length.

The Tagged Delimited technique uses delimiters to mark the end of the message element, whereas Tagged Fixed length elements have a known length (as in Fixed Length data element separation), and no delimiter is used.

It is possible for a single message to contain more than one type of data element separation, or to the use the same technique but with different properties (e.g. delimiter) throughout the message.

This SupportPac will demonstrate how to use the above techniques to model messages used in existing message standards, such as SWIFT, using the MRM message model found in WMQI.

To use Tagged/Delimited messages in WMQI, the Tagged/Delimited Wire Format Layer must first be added to the message set. Instructions to add format layers are available in the *Working With Messages Book* under *Adding wire formats* in the *Working with MRM messages in the Control Center* chapter. After adding the Tagged/Delimited Wire Format to the message set, a Tagged/Delimited related tab appears at the message set level. It is possible to set message set defaults here which will be used for all messages, unless they are overridden by the corresponding compound type properties.

# Installation

The SupportPac is supplied in a zip file, id05.zip. This should be uncompressed into the directory of choice, and includes the following files:

- msbeta.mrp — The example message set, containing the MRM models of messages used in this SupportPac.
- \Example_Data — Example message input files. See the Appendix for more details on the files contained.
- mqsiput.exe — Utility to put the example messages to an MQ queue.
- id05.pdf — This User Guide in Adobe Acrobat format.
- ID05MsgFlows.xml — Example message flows to process the messages.
- qdefs.dat — Queue definitions to be used with the example message flows

To use the example message set and messages provided with this SupportPac, the message set must first be imported into a Configuration Manager (see the appropriate *WebSphere MQ Integrator Installation Guide* for your platform).

1. Use mqsiimpexpmsgset to import the message set into the Message Repository database. For example:

   ```
   mqsiimpexpmsgset -i -n <MRM DB> -u <USERID> -p <PASSWORD> -f msbeta.mrp
   ```

2. Start the Control Center and on the Message Sets tab, right-click on Messages Sets folder, and use *Add to Workspace* to make the msbeta message set visible. Repeat this procedure for the Types, Elements, and Messages folders as required to view those objects.

3. Import the example message flows found in ID05MSGFlows.xml into your workspace, using the *Import to Workspace...* command.

4. Use the Webspher MQ commands program, runmqsc to define the queues to be used with the example message flows. A commands file has been provided to do this, using the command:

   ```
   runmqsc <QMGR> < qdefs.dat
   ```

5. Create the broker using the command line utilities, and then create it in the Control Center Topology. Assign and deploy the message flows and message set to it. Check the Log view for the result of the deploy operation. For more details see *WebSphere MQ Integrator Using the Control Center.*

Once the broker has been deployed successfully with the message set and flows it is ready to process the example messages used in this SupportPac.

# Chapter 2. Examples of Messages

## Creating Tagged/Delimited Format Messages

To enable WMQI to recognize and parse a Tagged/Delimited format message, it must first be defined in the Control Center. A *Compound Type* must be created to contain the elements of the message, and also a *Message* must be based on this compound type to allow the broker to use the definition.

The compound type is used to contain elements, and if the *Type Composition* of the compound type is *Sequence* or *Choice*, it may contain other compound types (see Control Center online help for more details on these options). These other compound types may have differing Tagged/Delimited format settings to the parent compound type, to allow more complex messages to be modeled.

## Delimited Messages

This section details how to create a message or part of a message which has *Delimiters* but does not have tags.

After initially checking in, the Compound Type will have a tab relating to Tagged/Delimited information. One of the attributes available is titled 'Data Element Separation'. When you check out the Compound Type, you will have a drop down containing a number of options for separation of elements.

Two of these options are suitable for modeling a message which has delimiters but not tags:

- All Elements Delimited
- Variable Length Elements Delimited

Use 'All Elements Delimited' if every element in the Compound Type is delimited.

Use 'Variable Length Elements Delimited' if some elements in the Compound Type are adjacent to each other and are not separated by a delimiter. If a non-zero length is specified in the Tagged/Delimited tab for an element, then it will be treated as a fixed length element, and will appear adjacent to the proceeding element. Otherwise the elements will be separated by the delimiter.

If you specify either of the above options, you must also specify the value of the delimiter in the field titled 'Delimiter', found on the same tab.

Only one Delimiter value can be specified per Compound Type. However, more than one delimiter can be used in a message by the use of nested Compound Elements[1] or additional Compound Types within the message.

Also, it is necessary to set any compound types using these Data Element Separation techniques to have a *Type Content* of "*Closed*". This is necessary as it is not possible to have self-defining elements in these types of messages.

To clarify the above, take the following example message:

- String 1 ending with Carriage Return, Line Feed.
- String 2 ending with Carriage Return, Line Feed.

[1] a nested Compound Element is an element based upon a compound type.

- Integer field 1 adjacent to Integer field 2, ending with Carriage Return, Line Feed.
- Integer field 3 adjacent to Integer field 4, ending with Carriage Return, Line Feed.
- String 3

The length of Integer field 1 MUST be known, otherwise we cannot know where Integer field 2 starts.

The length of Integer field 3 MUST be known, otherwise we cannot know where Integer field 4 starts.

The lengths of other fields may or may not be known.

Note that the elements that end with Carriage Return, Line Feed are delimited by those characters, and therefore the value of the Delimiter would be set to **<CR><LF>**.

The above can be modeled as a Compound Type with Data Element Separation set to 'Variable Length Elements Delimited', as some of the elements require a delimiter, whereas others do not. If it were the case that all elements ended in Carriage Return, Line Feed, then the Data Element Separation could be set to All Elements Delimited. The Compound Type for the above message would have the following elements defined:

- **String1**
- **String2**
- **Integer1**
- **Integer2**
- **Integer3**
- **Integer4**
- **String3**

The elements **Integer1** and **Integer3** must have their length defined in their Tagged/Delimited Format tabs.

As all other elements are delimited by <**CR><LF>**, no length should be specified for these.

In this SupportPac, the above message has been modeled. See messageset **msbeta**, message **Notagmess** for details. A messageflow to exercise the message (name **NotagMF1**) is also available. (For details on using the example data see Appendix A)

An example of the **Notagmess** message is given below:

```
XML<CR><LF>
blah<CR><LF>
00002
3<CR><LF>
004
5<CR><LF>
blahblah
```

The message flow is sensitive to the value of the **String1** field, and outputs in different wire formats dependent on this input field's value. In the example message above the message flow will output an XML message, as defined by the XML Format tab for the message. See Appendix A for more details on the example message flows.

**Integer1** is of fixed length, therefore the justification, padding character, and length properties found on its Tagged/Delimited Format tab have been used. In this case the element is 5 characters right-justified, using 0 as the padding character. These properties can be adjusted as required - see Control Center OnLine help for more details. **Integer3** is similar, but only 3 characters long.

The final string element, **String3**, does not have a delimiter after it as it is the last element in the message. If characters were required to follow this last element, then they would be defined in the Group Terminator field. See Control Center online help for more information on the Group Terminator and other Tagged/Delimited settings.

## Tagged/Delimited Messages

This section details how to create a message or part of a message which has delimiters and tags. It concludes with example definitions of MT100 and MT103 messages from the SWIFT standard.

As for Delimited Messages, a Compound Type must be created which will contain the elements, and other compound types as required. It is also possible to base elements on compound types, which allows nested structures to be incorporated into the message, as elements have tags, whereas a compound type does not.

After initially checking in, the Compound Type will have a tab relating to Tagged/Delimited information. One of the attributes available is titled 'Data Element Separation'. When you check out the Compound Type, you will have a drop-down containing a number of options for separation of elements. One of these options is suitable for modeling a message which has delimiters and tags:

- Tagged Delimited

For Tagged Delimited messages it is necessary to specify a Delimiter value, and also either a Tag Data Separator, or a Tag Length, which were described in the Overview section. For each element in the Compound Type, you must also assign a Tag value. You need to check out each element to do this.

### SWIFT

Each SWIFT message consists of several headers, a mandatory basic header, and several other optional elements, as outlined below.

```
{1: BASIC HEADER}
{2: APPLICATION HEADER}
{3: USER HEADER}
{4: MESSAGE BODY}
{5: TRAILER}
```

Each block begins with a ‘{’ character, and ends in a ‘}’, and contains a tag (1-5) separated from the data by a colon. The message body may also contain sub-blocks of fields, this is the section that varies most between messages, and will need the most attention when creating a model of the message. The headers and trailers are fairly constant between the different messages.

In the Supportpac, SWIFT MT100 and MT103 messages have been modeled, and will be described below. See messageset **msbeta**, messages **MT100** and **MT103** for details.

**MT100**

The MT100 message, as set out by the SWIFT standards, contains a variety of mandatory, and optional elements. Some elements can also be one of several different types, e.g. the fourth element can either be field element 52A or 52D, which take on different structures. The message elements may also contain a structure within, e.g. field 32A contains date, currency type, and amount information, and it is possible to model these components individually.

To clarify the above, the mandatory components of a SWIFT MT100 message are illustrated below:

Example (SWIFT MT100 message):

```
{1:F01BANKBEBBAXXX2222123456}{2:I100IBMADEF0AXXXN}{3:{108:abc}}{4:
:20:X
:32A:940930USD1000000,
:50:LINE1
:59:LINE1
-}
```

The example message comprises three headers and a message body. Some SWIFT messages may also contain a trailer.

Each header, body and trailer is wrapped within curly braces ({...})

The message contains a number of mandatory Carriage Return and Line Feed characters. To illustrate, the above message is repeated below, with <CR><LF> added to indicate where the Carriage Return and Line Feed characters are.

```
{1:F01BANKBEBBAXXX2222123456}{2:I100IBMADEF0AXXXN}{3:{108:abc}}{4:<CR><LF>
:20:X<CR><LF>
:32A:940930USD1000000,<CR><LF>
:50:LINE1<CR><LF>
:59:LINE1<CR><LF>
-}
```

The following can be deduced from the above example:

- The first header is identified by the 1 after the open curly brace, '{', and a colon follows it, the header is terminated by a close curly brace, '}'.

- The second header is identified by the 2 after the open curly brace, '{', and a colon follows it, the header is terminated by a close curly brace, '}'.

- The third header, the user header, is identified by the 3 after the open curly brace, '{', and a colon follows it. The header may contain other structures within it, again identified by open and close curly braces. The user header is terminated by a close curly brace, '}'.

- The message body is identified by the 4 after the open curly brace and a colon follows it. Each element is presented on a new line for clarity. Note that the elements are separated by **<CR><LF>:**, and the tag and data are separated by a colon. The block is terminated with **<CR><LF>-}**. Therefore, the message body delimiter is **<CR><LF>:**, the Tag Data Separator is colon, and the Group Terminator is **<CR><LF>-** (The final close curly brace is the Group Terminator for the overall message).

- The message body has a number of colons which surround the Tags. Within the example message body, 20 32A 50 and 59 are Tags, these will need to be assigned to the corresponding message elements, and also Tagged Delimited will need to be set as the Data Element Separation technique.

Within the overall message, it is possible to model the header and body identifiers as Tags (that is, values 1, 2, 3 and 4 mentioned above). However, problems may occur because it is likely that each SWIFT message body would be modeled as a separate Compound Element/Compound Type and as such each would need to be allocated a Tag value of 4. This is an issue, as within a message set, no two elements can have the same Tag value.

It is possible to circumvent this problem by setting the message body Compound Type Group Indicator to include the tag, e.g. **4:<CR><LF>:**. When creating the overall message compound type, (i.e. the compound type that the entire message is based upon), Variable Length Elements can be used. The Compound Type Group Indicator and Terminator are set to open and close curly braces to represent the first and last characters in the message. The delimiter is set to closed and open curly braces, as these are the characters between elements in the message.

Note that a future release of this SupportPac, planned to be released early next year, will model the overall message structure slightly differently.

So a possible model for the overall message **MT100** which contains the headers, trailer and message body would be:

| | |
|---|---|
| Compound Type Group Indicator | { |
| Compound Type Group Terminator | } |
| Compound Type Data Element Separation | Variable Length Elements Delimited |
| Compound Type Delimiter | }{ |

The first two headers in the sample message could be defined either as simple STRING elements, or as Compound Elements based upon Compound Types that contain elements for each of the header components, and would have Data Element Separation set to 'Fixed Length'. Within the Compound Type, each element would then need a length set in its Tagged/Delimited tab. In the example message set the headers have been modeled using STRING elements, and the whole header is treated as one block of data.

The third header (3) has to be modeled as a compound element. This is because the curly braces contained within the message will otherwise be interpreted as delimiters for the main message. A compound element is created, using Tagged Delimited as the Data Element Separation technique, and elements are created according to the specification (see type **t_USERHEADER** for details). The header compound element, **e_USERHEADER**, is created from a compound type containing the **t_USERHEADER** compound type, and also a string element used to contain the tag of 3 that identifies the header.

The example overall message is called **MT100**, which contains the **BASICHEADER** string, the **APPLICATIONHEADER** string, the **e_USERHEADER** compound element, the **MT100 Body** compound element, and the **TRAILER** compound element. The **MT100 Body** element is based upon a Compound Type **Type_MT100**, which models the message body of an MT100 message, using the settings below:

| | |
|---|---|
| Compound Type Group Indicator | 4:<CR><LF>: |
| Compound Type Group Terminator | <CR><LF>- |
| Compound Type Tag Data Separator | : |
| Compound Type Data Element Separation | Tagged Delimited |
| Compound Type Delimiter | <CR><LF>: |

As mentioned previously, the field **32A** contains a substructure, and this is reflected in the supplied MT100 model. The element has a tag of **32A**, but then contains a structure within (click on the + to show the sub-elements), and the element is based upon **Type_T32A**. If this compound type is viewed, it is possible to see that it uses the Variable Length Elements Delimited data element separation technique. The first two elements are of fixed length, whereas the final element is variable length, hence the delimiter from **Type_T32A** is not used (although it has to be specified). The delimiter from the **Type_MT100** compound type will appear after the element though, as the **32A** element is delimited due to it's variable length. As the constituent parts of field 32A have been modeled, this allows them to be referenced individually by ESQL.

The granularity to which the element is modeled is determined by how the information contained within is to be used. If the message field **32A** was not to be used anywhere else in the message flow, then it could have been modeled by using a simple STRING element.

The MT100 message also contains elements with a choice of structures. These structures are modeled by using *choice* types, and within these are contained elements with different structures. Only one of the elements in the choice type, is used for any one message. In the case of field 52, options A or D may be used with message MT100. Opening up the **Type52_Choice** type shows the two options possible. Also, element **52A** demonstrates the use of different data element separation techniques used in the same element, as it contains both a Variable Length Elements Delimited section (**Party Identifier**) and a Fixed Length section (**BIC_Type**). It is possible to add compound types into both the **Type_MT100**, and **BIC_Type** compound types, as they are of Type Composition Sequence, as mentioned previously.

**MT103**

The MT103 SWIFT message is similar to the MT100 in structure, but also contains optional repeating elements. These are modeled in WMQI using the repeat property of the element's Connection tab, see the Control Center Online help for more details on this property.

**Example Data**

Messageflows (**MT100MF1** and **MT103MF1**) to exercise the SWIFT messages described above are available in the Supportpac, along with sample data. See Appendix A for details.

# Appendix A - How to Use the Example Data

Three message flows are provided with this SupportPac:

1. NotagMF1 - for use with the **Notagmess** message definition.

2. MT100MF1 - for use with the **MT100** message.

3. MT103MF1 - for use with the **MT103** message.

The files found in the \Example_Data folder contain example messages for the above flows. There are folders relating to each of the messages modelled, and within these folders are the data files to be used with the **mqsiput.exe** program. The files follow the naming convention:

<message_name><input_format><output_format>.txt

The messageflows are sensitive to the value of the element **20**, or **String1** (depending on the message) and output in different wire formats dependent on this input field's value.

| Value of field **20** / **String1** | Output Format |
|---|---|
| SWIFT | SWIFT, in the same codepage as input |
| EBCDIC | SWIFT, in codepage 37 |
| CWF | Custom Wire Format, as defined in the message |
| XML | XML, as defined in the message. |

N.B. When using CWF for output, all choices must be specified, as the CWF writer will fail otherwise. This is because it will not decide which choice to write out if none is specified. Also, if you intend to use CWF for output, and don't always expect to have all fields filled in, then it is necessary to specify default values for those elements.

A Windows NT/2000 command line utility, **MQSIPUT.EXE** has been provided with this SupportPac to allow the sample messages to be placed on an MQ queue. Full details on this utility can be found in SupportPac IH02.

Usage: mqsiput <QNAME> <QMGR> <FILENAME>

Other standard MQ utilities, such as the MQSeries Explorer can be used to view the output messages.

End of Document