MQSeries Integrator V2 - XML to MRM Utility

Version 1.3

20 November 2001

William A. Matthews, Jr.
IBM Dallas Systems Center
7 Campus Circle
Roanoke, TX 76272
USA

cicsos2@us.ibm.com

Take Note!

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

Second Edition, November 2001

This edition applies to Version 1.3 of *MQSeries Integrator V2 – XML to MRM Utility* and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Notices

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

Information contained in this report has not been submitted to any formal IBM test and is distributed “AS-IS”. The use of this information and the implementation of any of the techniques is the responsibility of the reader. Much depends on the ability of the reader to evaluate these data and project the results to their operational environment.

The performance data contained in this report was measured in a controlled environment and results obtained in other environments may vary significantly.

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
- MQSeries Integrator
- MQSI

The following terms are trademarks of other companies:

- Windows NT, Windows 2000 Microsoft Corporation

Summary of Amendments

Date	Changes
25 Sept, 2000	Initial release
20 Nov, 2001	Additional support for XML Attributes and comments added. Generated structures now have field length set to 1 to avoid duplicate fields. The sample solution has been extended and is now available as part of the SupportPac.

Preface

This SupportPac provides a utility, written in REXX, that is used to create a C structure or COBOL copybook from an XML structure that can then be imported into the MRM repository using the MQSeries Integrator Control Center. The imported definition can then be used within a Database or Compute node in a MQSI message flow. This allows the drag and drop capabilities of these nodes to be utilized, thus reducing the time to complete the ESQL coding.

Bibliography

List any supporting publications here otherwise delete this page. Use following format:

- *The REXX Language, A Practical Approach to Programming,* Pub. Prentice Hall, Inc. by M.F. Cowlishaw

- *MQSeries Integrator Using the Control Center V2.0.1*, IBM Corporation. SC34-5602

Chapter 1. Overview

IBM's MQSeries Integrator (MQSI) provides a powerful solution to the challenge of formatting and reformatting data. In its simplest form, MQSI 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 MQSI 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. Two of the MQSI provided formats are addressed by the XML2MRM utility. Specifically these are XML and C or COBOL definitions (MRM). Additional background information is discussed in Chapter 3 of *MQSeries Integrator Using the Control Center V2.0.1*. From a high level view, the basic function of the XML and MRM formatters (also called parsers) is to analyze a message and convert it into a "tree" of fields. The XML formatter constructs the tree based on the XML structure and tag names. Thus, a well-formed XML message can be parsed without requiring additional descriptive information. The MRM formatter must have a specific, field by field, definition of the message. This definition is based on either a C language structure or COBOL copybook. The MRM does impose some additional restrictions on these definitions. These considerations are discussed in more detail in the Using the Control Center manual. Some important points to consider are:

- Although XML tags may use a dash (**-**) as a word separator, this is not valid with an MRM definition and must be replaced with an underscore (_).

- The list of reserved words is also documented in the Using the Control Center. It includes both C language and COBOL as well as ESQL reserved words.

- When a COBOL copybook is used to map a XML message tree, the XML tags must also be in upper case. This is not true when a C structure is used.

The purpose of the XML2MRM utility is to provide a means whereby an XML message can be converted into a C language structure or COBOL copybook. This will allow that definition to be imported into the MQSI Message Repository Manager (MRM) and then to be used as part of the drag and drop functions in the Message Flow Compute and Database nodes.

There are a few considerations when using this mapping approach:

- XML tag names must meet the naming considerations of the MRM. For example, a dash (-) is valid in an XML tag, but is not valid in an MRM field name. Thus, when designing XML tag names, an underscore (_) must be used instead. In addition, tag names cannot duplicate an MRM reserved word. The C, COBOL importer will identify XML tag names that are not acceptable. The import utility report should always be reviewed when importing a new definition. For example, the Sales Order Sample uses Item as an XML tag name. Since this is a reserved word, it must be coded as "Item" in the ESQL statements.

- The XML2MRM utility will set all field length values to 1. It is the user's responsibility to ensure that the generated length information is valid and does not exceed the target field length.

- The XML2MRM utility is not designed to handle highly complex XML structures. However, it now supports XML structures that make use of attributes within a tag.

- There is no attempt to provide special handling for repeating fields. These require additional manual work on the generated result. The sample solution has been expanded to show the use of repeating (fixed count) fields.

- It is assumed that the utility will be run from a command prompt and that fully qualified path and file names will be used.

- When the input message is XML and the output message is MRM, the EQSL code built with the Compute node's drag and drop can be used unchanged. The Message Domain property of the MQInput Node must be set to XML.

- *Although not shown in the sample solution or the following description, the parser type for the message set may be changed from MRM to XML and the generated statements will be correct.*

  The following is an example of a generated statement:

```
SET "OutputRoot"."MRM"."B1"."VN" = InputBody"."MEMBER_PROFILE_REQUEST"."B1"."VN";
```

- When the input message is MRM and the output message is XML, then the MessageDomain and MessageFormat must be changed. The same is true for XML to MRM. An example of the changes is shown in the second example below.

    - The generated SET statements for the MessageSet and MessageType values do not need to be changed.

```
SET OutputRoot.Properties.MessageSet = 'DHSHPGG098001';
SET OutputRoot.Properties.MessageType = 'm_My_Message'
```

    - However, the MessageDomain and MessageFormats settings will be copied from the input message and will have to be changed, as shown below:

```
SET OutputRoot.Properties.MessageDomain = 'XML'
Or
SET OutputRoot.Properties.MessageDomain = 'MRM'
SET OutputRoot.Properties.MessageFormat = 'XML';
Or
SET OutputRoot.Properties.MessageFormat = 'CWF';
```

    - Finally, the generated SET statements must be modified. The "MRM", as shown in the first line, must be changed to "XML". A good technique is to save a copy of the modified code in a text file and then remove the drag and drop definitions from the first page of the Compute node. Finally replace the ESQL statements with the saved version (placing them after the "-- Set" separator provided in V2.0.1).

```
SET "OutputRoot"."MRM"."MEMBER_PROFILE_RESPONSE"."B1"."VN" = "InputBody"."B1"."VN";
SET "OutputRoot"."XML"."MEMBER_PROFILE_RESPONSE"."B1"."VN" = "InputBody"."B1"."VN";
```

Chapter 2. Installation and Usage

Prerequisites

The SupportPac requires:

- MQSeries Integrator V2
- Windows NT or Windows 2000
- A REXX Interpreter. The following implementations has been tested:
  - IBM ObjRexx - http://www.software.ibm.com/ad/obj-rexx
  - Regina Rexx - http://www.lightlink.com/hessling/Regina

Installation

The SupportPac is delivered as a single file in a standard zip format. When the file is unzipped, several files should be produced, as follows:

- ia04.pdf This User Guide in Adobe PDF format
- license2.txt License file
- xml2mrm.rex The utility
- \sample Sample sub-directory

Place the **xml2mrm.rex** code into a suitable directory. This is the only executable part of the SupportPac.

Execution

The method to run the XML2MRM utility depends upon the REXX implementation installed. From a command prompt:

Regina Rexx	**IBM ObjRexx**
Regina xml2mrm input output type	Rexx xml2mrm input output type

Where:

Input is the name of the file containing the XML message to be analyzed

Output is the name of the file to contain the generated results

Type is C (default) or COBol

When the input and/or output files are in different directories, full path information must be provided. If the REGINA or REXX drivers cannot find the xml2mrm.rex file, you will need to provide full path information there also.

Chapter 3. Sample Sales Order Example

Background

This application collects sales order information from a Websphere Application Server. The information is formatted into the Sales Order XML structure. The XML message is sent to MQSI where it is reformatted into a COBOL bitstream and passed to the CICS server application. Confirmation is returned in a similar fashion. For purposes of this example, the COBOL structures and XML messages are identical.

In addition, a sample solution, including a message set import file (sales_order.mrm) and a message flow import file (IQA04 Sample Sales order_export.xml).

Figure 1 is the Sales Order XML structure and the maximum size of each field is indicated by a series of “x”. **Figure 2** is the Sales Order COBOL structure. Note that there are repeating fields defined. **Figure 3** is the C structure generated by the XML2MRM utility. **Figure 4** shows that a manual change was made for the repeating fields.

The remaining 10 figures are screen images of the process used to import the C structure and COBOL copybook into a single message set. Once the messages were created, the next step was to create a pair of message flows. These two flows are shown together (this technique does work) in **Figure 14**. The flow that processes the XML input and generates COBOL output is shown first. Of particular interest is the MQInput Node properties shown in **Figures 6** and **9** for the request and reply flows. In **Figure 6**, the message format and domain are shown as XML. **Figures 7** and **8** show the status of the Compute Node screen after the drag and drop operation was completed and the generated ESQL code. This generated code is used without change.

In **Figure 9**, since the message does require MRM formatting, related details have been filled for the example. **Figures 10** and **11** again show the drag and drop status and generated ESQL for the Compute Node. However, unlike the first message flow (XML to MRM), this message flow (MRM to XML) **will require** modification. In **Figure 12**, the generated ESQL code has been copied into a notepad and edited. The following actions were taken:

- The line “-- Enter SQL below this line.” Has been moved above the SET statements that were generated by the drag and drop.
- The series of SET "OutputRoot".MRM." … lines have been modified to SET "OutputRoot".XML." This identifies the output as XML.
- The following two lines must also be added to complete the transformation:

  SET OutputRoot.Properties.MessageDomain = 'XML';

  SET OutputRoot.Properties.MessageFormat = 'XML';

Note that the entire process took less than an hour.

Samples

```
<SalesOrder>
    <Custno>xxxxxxxxxx</Custno>
    <Suffix>xxx</Suffix>
    <PONum>xxxxxxxxxxxxxxxxxxxxxxxxx</PONum>
    <CommitFlag>x</CommitFlag>
    <Item>
       <ItemNo>xxxxxx</ItemNo>
       <Qty>xxxxxx</Qty>
       <NeedBy>xxxxxxxx</NeedBy>
    </Item>
    <ShipVia>xxxx</ShipVia>
    <PmtMeth>xxxx</PmtMeth>
    <CCNo>xxxxxxxxxxxxxxxx</CCNo>
    <ExpDate>xxxx</ExpDate>
    <Contact>xxxxxxxxxxxxxxxxxxxxxxxxx</Contact>
    <Phone>xxxxxxxxxxxx</Phone>
    <MsgLine>
        <Msg>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</Msg>
    </MsgLine>
</SalesOrder>
```

Figure 1. Sales Order XML Structure – showing maximum lengths of each field

```
       01 SALES-ORDER-DATA.
         05 SM-CUSTNO             PIC X(10).
         05 SM-SUFFIX             PIC X(03).
         05 SM-PONUM              PIC X(25).
         05 SM-COMMIT             PIC X(01).
         05 SM-ITEM OCCURS 20 TIMES INDEXED BY ITEM-IDX.
           10 SM-ITEMNO           PIC X(06).
           10 SM-QTY              PIC X(06).
           10 SM-NEEDBY           PIC X(08).
         05 SM-SHIPVIA            PIC X(0004).
         05 SM-PMTMETH            PIC X(0004).
         05 SM-CCNO               PIC X(0016).
         05 SM-EXPDATE            PIC X(0004).
         05 SM-CONTACT            PIC X(0025).
         05 SM-PHONE              PIC X(0012).
         05 SM-MSGLINE OCCURS 10 TIMES INDEXED BY NMSG-IDX.
           10 FILLER              PIC X(50).
```

Figure 2. Sales Order COBOL Structure

```
struct SALESORDER_xml2mrm {
  struct SalesOrder {
        char Custno[10];
        char Suffix[3];
        char PONum[25];
        char CommitFlag[1];
   struct Item_t {
         char ItemNo[6];
         char Qty[6];
         char NeedBy[8];
   } Item;
        char ShipVia[4];
        char PmtMeth[4];
        char CCNo[16];
        char ExpDate[4];
        char Contact[25];
        char Phone[12];
   struct MsgLine_t {
         char Msg[50];
   } MsgLine;

   } SalesOrder;
} SALESORDER;
```

Figure 3. Sales Order C structure generated by XML2MRM

```
struct SALES_ORDER_xml2mrm {
  struct SalesOrder {
        char Custno[10];
        char Suffix[3];
        char PONum[25];
        char CommitFlag[1];
   struct Item_t {
         char ItemNo[20][6];
         char Qty[20][6];
         char NeedBy[20][6];
   } Item;
        char ShipVia[4];
        char PmtMeth[4];
        char CCNo[16];
        char ExpDate[4];
        char Contact[25];
        char Phone[12];
   struct MsgLine_t {
         char Msg[10][50];
   } MsgLine;

   } SalesOrder;
} SALES_ORDER;
```

Figure 4. Sales Order C structure modified for repeating fields

Request to Server Message Flow Details

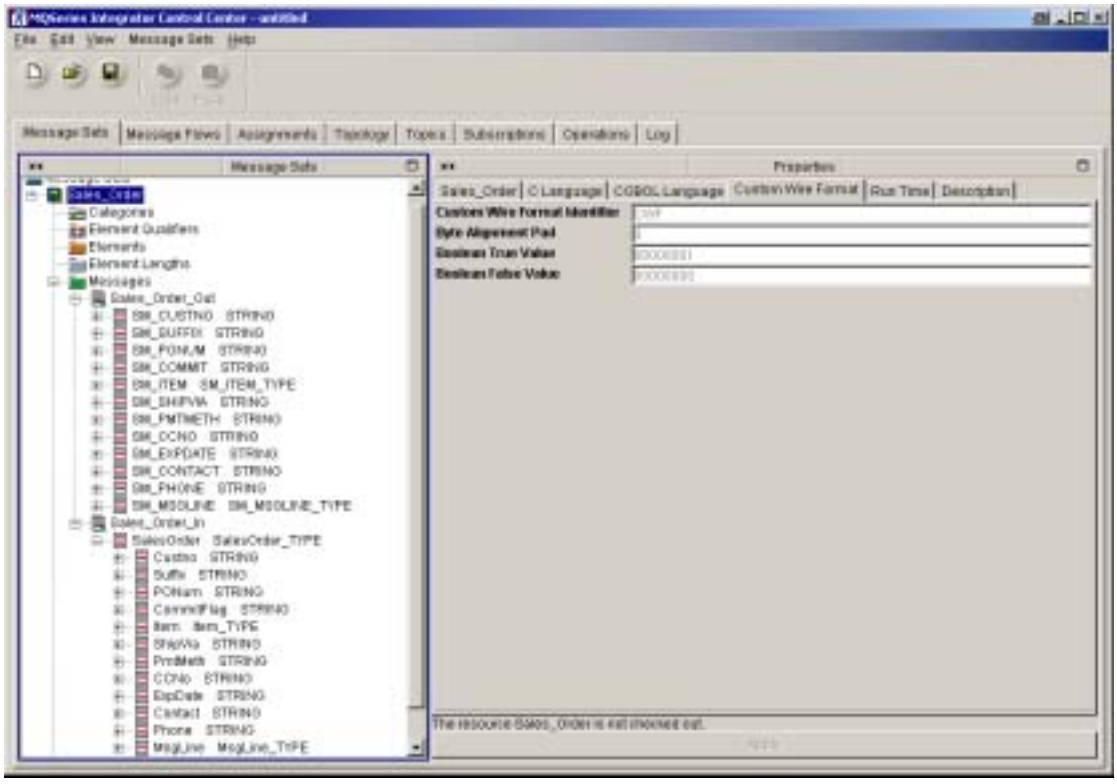

Figure 5. Results of MRM Import and Message Set creation

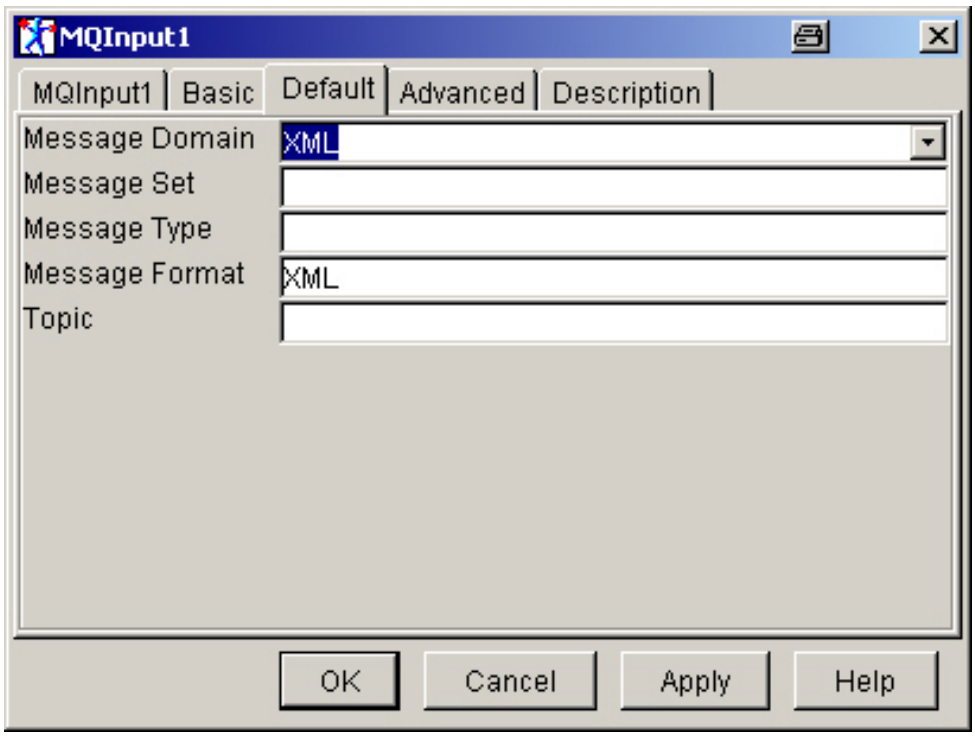

Figure 6 – MQInput Node Properties

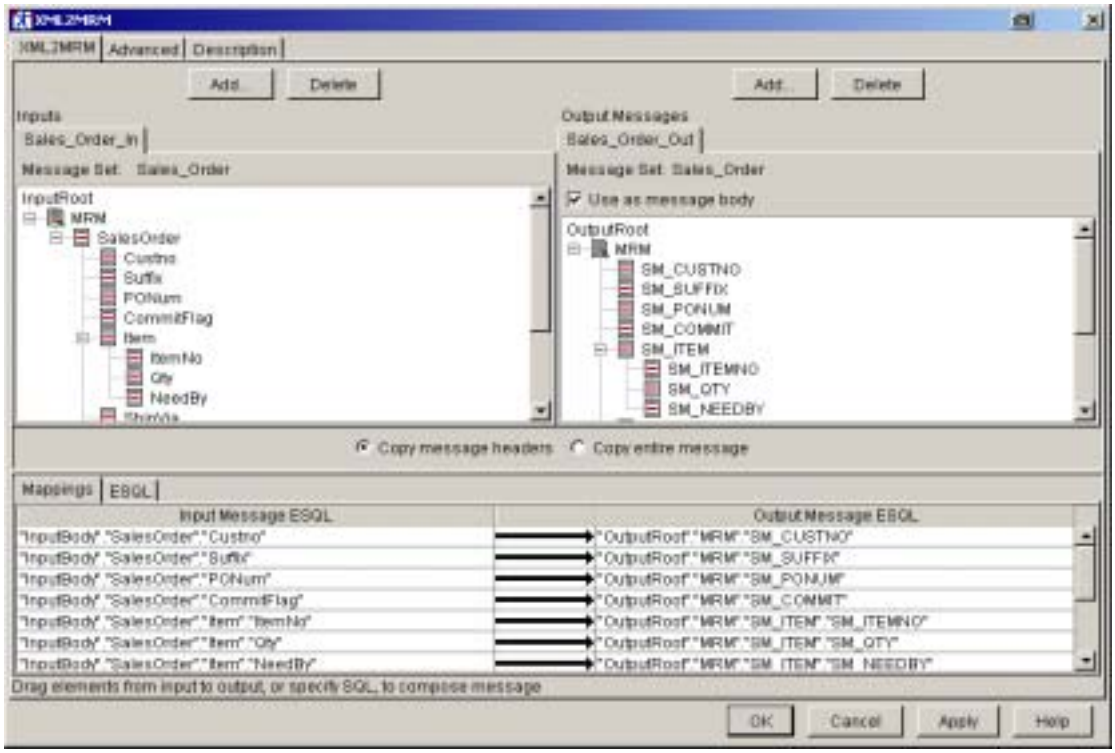

Figure 7. Compute Node (XML2MRM) Drag and Drop

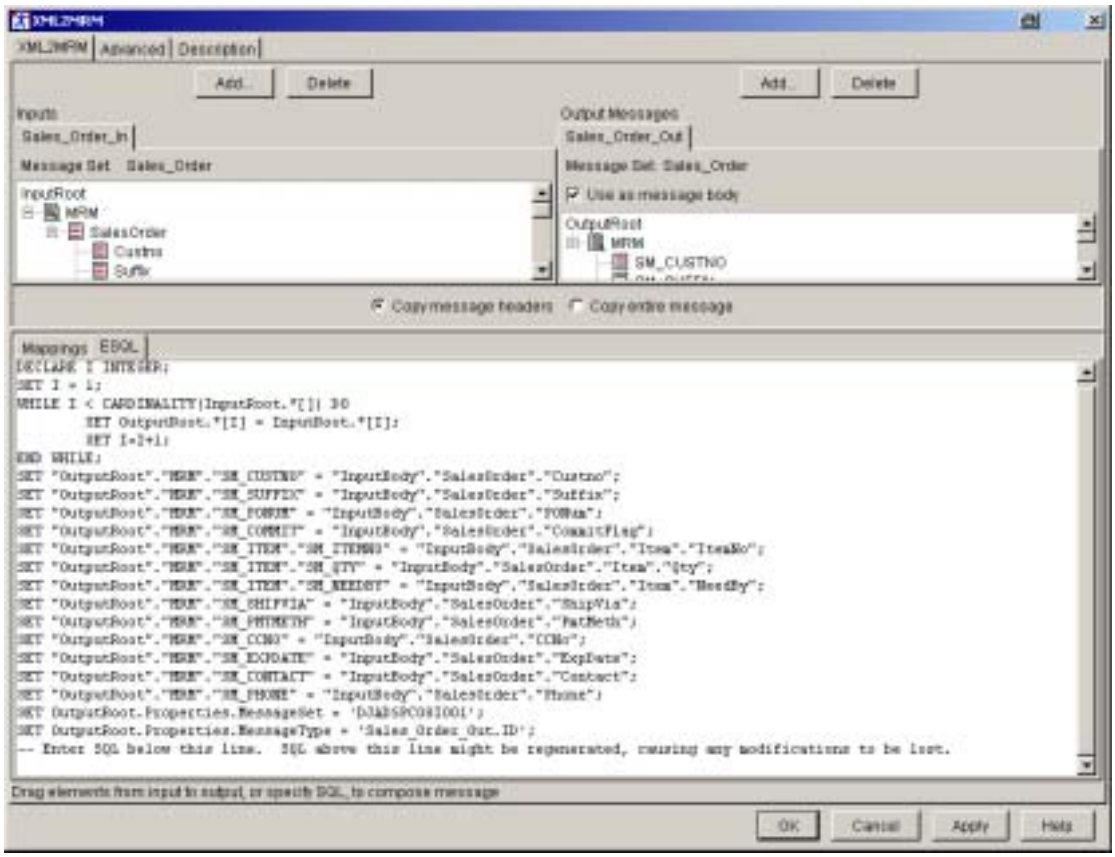

Figure 8. Generated ESQL Code – Used without changes

Reply from Server - Message Flow Details

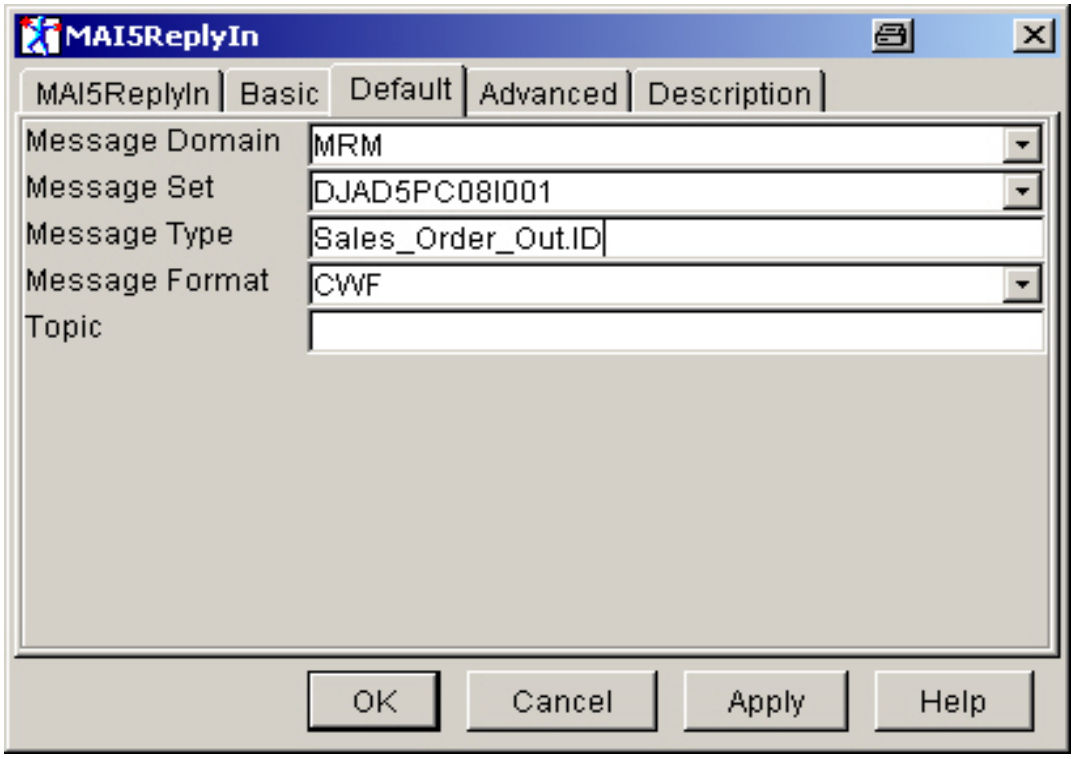

Figure 9. MQInput Node for Reply Message Properties

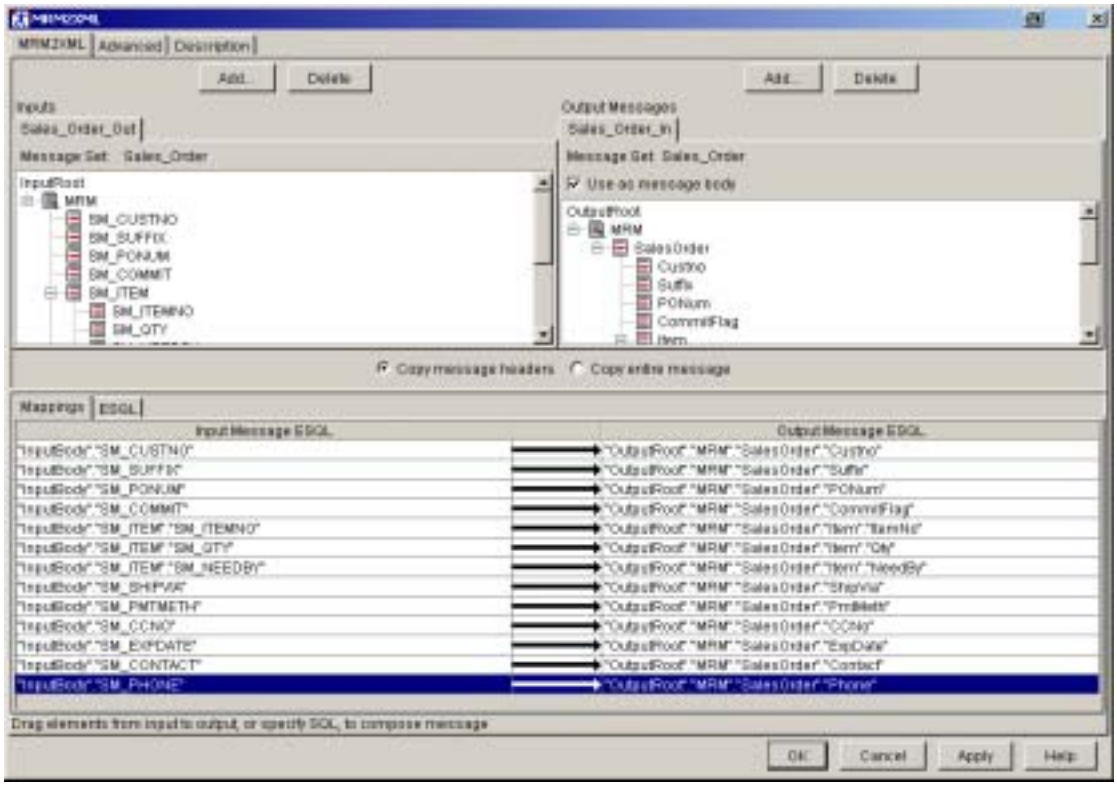

Figure 10. Compute Node (MRM2XML) Drag and Drop

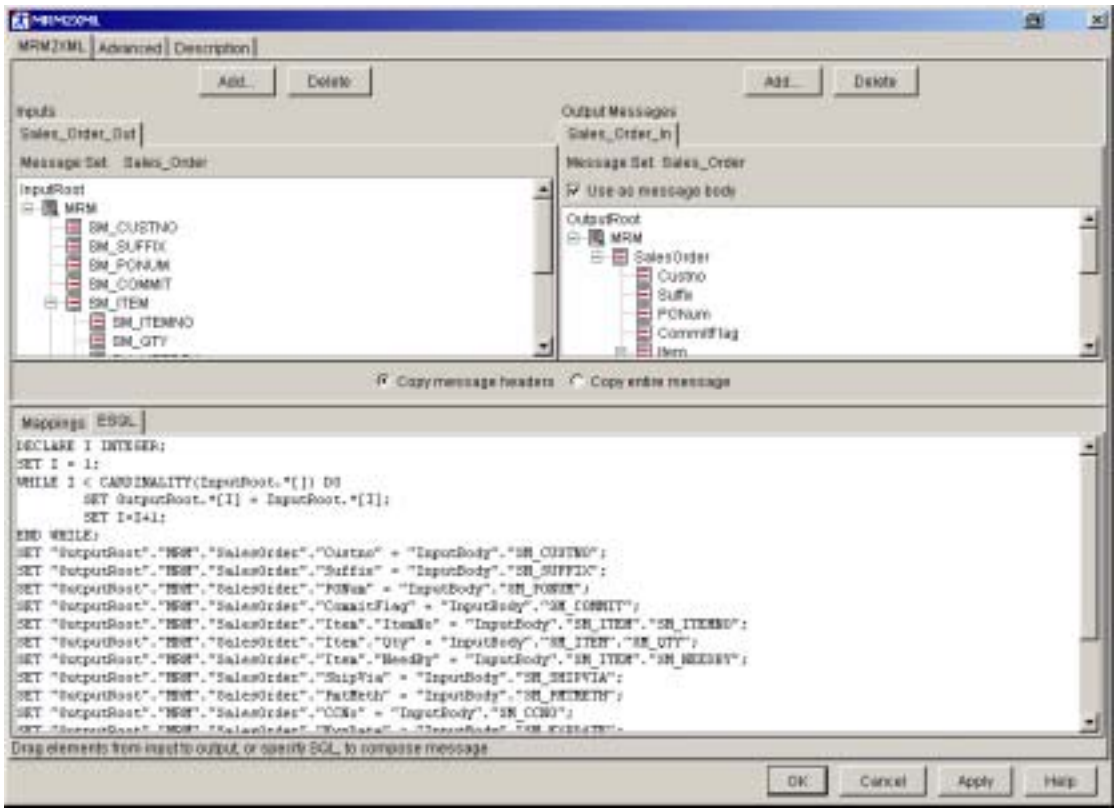

Figure 11. Generated ESQL Code – Must be Modified

```
DECLARE I INTEGER;
SET I = 1;
WHILE I < CARDINALITY(InputRoot.*[]) DO
       SET OutputRoot.*[I] = InputRoot.*[I];
       SET I=I+1;
END WHILE;
-- Enter SQL below this line.  SQL above this line might be regenerated,
causing any modifications to be lost.
/*_____________________________________________________________________*/
/* The generated code has been moved below the separator line to      */
/* accidental loss.  The original "MRM" has been changed to XML      */
/*_____________________________________________________________________*/

SET "OutputRoot".XML."SalesOrder"."Custno" = "InputBody"."SM_CUSTNO";
SET "OutputRoot".XML."SalesOrder"."Suffix" = "InputBody"."SM_SUFFIX";
SET "OutputRoot".XML."SalesOrder"."PONum" = "InputBody"."SM_PONUM";
SET "OutputRoot".XML."SalesOrder"."CommitFlag" = "InputBody"."SM_COMMIT";
SET "OutputRoot".XML."SalesOrder"."Item"."ItemNo" =
"InputBody"."SM_ITEM"."SM_ITEMNO";
SET "OutputRoot".XML."SalesOrder"."Item"."Qty" =
"InputBody"."SM_ITEM"."SM_QTY";
SET "OutputRoot".XML."SalesOrder"."Item"."NeedBy" =
"InputBody"."SM_ITEM"."SM_NEEDBY";
SET "OutputRoot".XML."SalesOrder"."ShipVia" = "InputBody"."SM_SHIPVIA";
SET "OutputRoot".XML."SalesOrder"."PmtMeth" = "InputBody"."SM_PMTMETH";
SET "OutputRoot".XML."SalesOrder"."CCNo" = "InputBody"."SM_CCNO";
SET "OutputRoot".XML."SalesOrder"."ExpDate" = "InputBody"."SM_EXPDATE";
SET "OutputRoot".XML."SalesOrder"."Contact" = "InputBody"."SM_CONTACT";
SET "OutputRoot".XML."SalesOrder"."Phone" = "InputBody"."SM_PHONE";
SET OutputRoot.Properties.MessageSet = 'DJAD5PC08I001';
SET OutputRoot.Properties.MessageType = 'Sales_Order_In.ID';
/*_____________________________________________________________________*/
/*        Reset  Properties to be XML                                  */
/*_____________________________________________________________________*/
SET OutputRoot.Properties.MessageDomain = 'XML';
SET OutputRoot.Properties.MessageFormat = 'XML';
```

Figure 12. Generated ESQL Code after modification

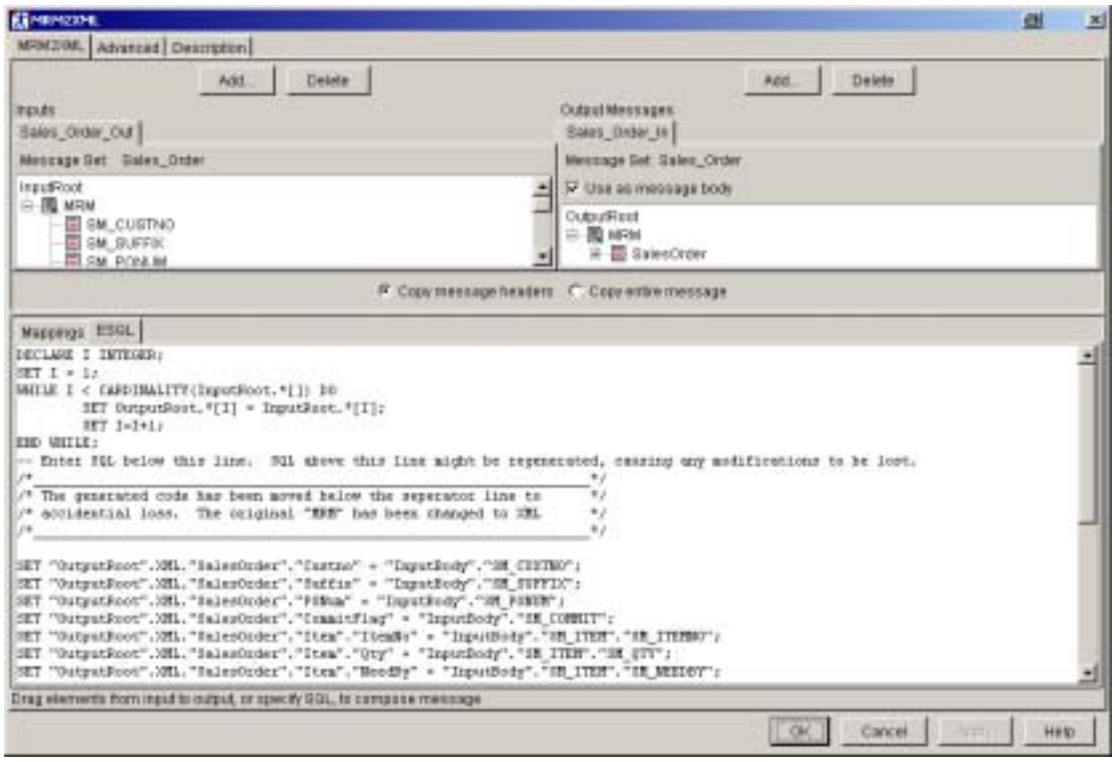

Figure 13. Modified Code Display

Message Flow Screen – Request and Reply

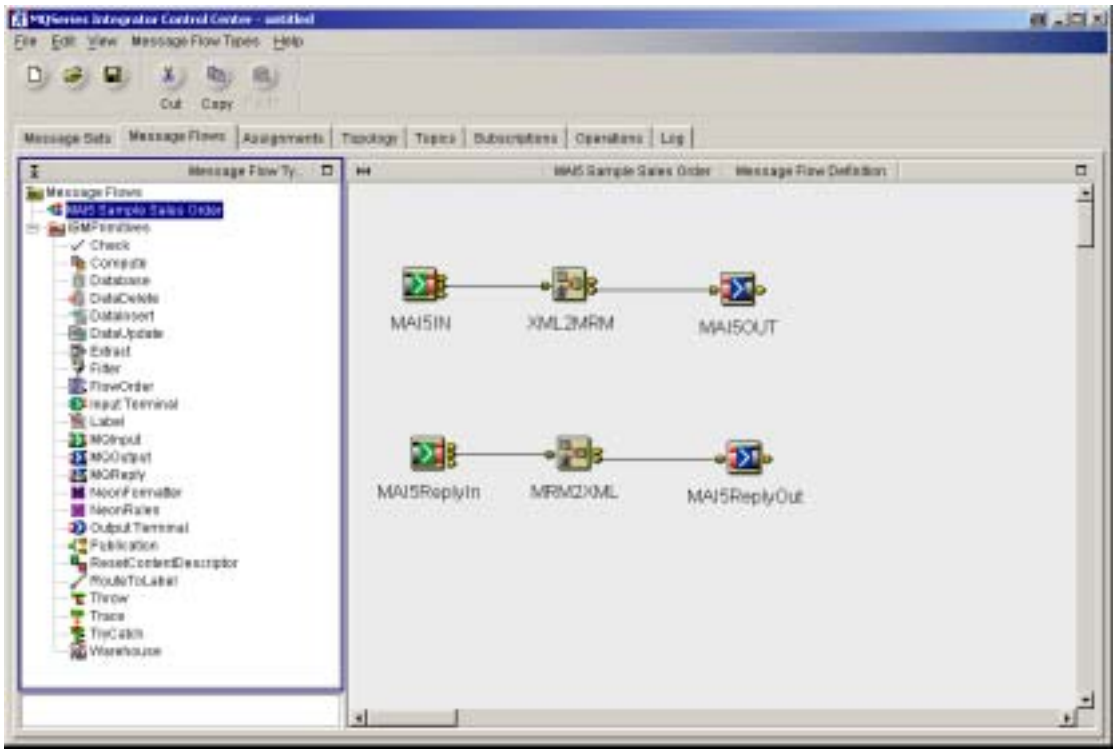

Figure 14. Message Flow – Request and Reply

Chapter 4. Lessons Learned

- When mapping XML via COBOL definitions imported using the C/Cobol importer, the XML tag names must be upper case and any dashes must be replaced with underscores.

- The first XML tag must be represented within the COBOL structure by inserting the same name following the 01 level and it must be higher than any existing levels. A similar technique is used for C structures by wrapping the definition with a new name.

- Always run a report first when importing a data definition into the MRM and check both the top and bottom of the report for errors. A common problem is to have a field name that is reserved.

- Establish naming conventions for messages sets and associated message flows.

- When using the drag and drop capabilities of the Compute Node to build a XML output, change the generated code to specify **XML** rather than **MRM**.

- In general, the steps to building a message set via the import process are:

    1. Create the message name (right click on **Message Sets**)

    2. When the new message is created, make note of the DHxxxxxxxx id value.

    3. For the C/COBOL import, first run a report and check the results, then do the real import.

    4. After the import, right click on **Types** and select *Add to Workspace* and *Compound Types.*

    5. Select the compound type that includes the entire message.

    6. Right click on **Messages** and select the (second on the list) *Create* option.

    7. The last steps are to name the message set and its identifier (suggestion is to use “m_” as the first two characters of the id or “.ID” as the last three and cut’n’paste the name to the rest of the id field. Also, associate the message to the message type.

- If any of the default values for a message must be changed, for example repeating and/or non-mandatory fields) this is a good type to do so.

- A field definition and associated compound types associated will have to be checked out before a field attribute can be changed.

- Always check the bottom line of the field properties display to see if any other Type value needs to be checked out.

- Always select *Apply* (on the last line) before changing to another field or another folder for the same field. Always reverify that the change was effective.

- When writing looping ESQL code, be careful to not have a situation where a loop is never ending. The DataFlowEngine that is running the message flow will absorb almost all of the remaining machine cycles and can be difficult to stop, even from the Operations Tab on the Control Center until it runs out of stack space.

End of Document