WebSphere Message Broker v7.0

IA9Y: WebSphere Message Broker Toolkit Map to ESQL PlugIn

Version 2.0
January 2010

Ben Thompson

IBM Software Services for WebSphere
IBM UK Laboratories
Hursley Park
Winchester
Hampshire
SO21 2JN

Please take Note!

Before using this report, please be sure to read the paragraphs on “disclaimers”, “warranty and liability exclusion”, “errors and omissions”, and the other general information paragraphs in the "Notices" section below.

Second Edition, January 2010.

This edition applies to WebSphere Message Broker V7 (and to all subsequent releases and modifications until otherwise indicated in new editions).

Notices

DISCLAIMERS
The performance data contained in this report were measured in a controlled environment. Results obtained in other environments may vary significantly.

You should not assume that the information contained in this report has been submitted to any formal testing by IBM. Any use of this information and implementation of any of the techniques are the responsibility of the licensed user. Much depends on the ability of the licensed user to evaluate the data and to project the results into their own operational environment.

WARRANTY AND LIABILITY EXCLUSION
The following paragraph does not apply to the United Kingdom or any other 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 NON-INFRINGEMENT, 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. In Germany and Austria, notwithstanding the above exclusions, IBM's warranty and liability are governed only by the respective terms applicable for Germany and Austria in the corresponding IBM program license agreement(s).

ERRORS AND OMISSIONS
The information set forth in this report could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; any such change will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time and without notice.

INTENDED AUDIENCE
This report is intended for architects, systems programmers, analysts and programmers wanting to convert WebSphere Message Broker V7 mapping files into ESQL code.

LOCAL AVAILABILITY
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. Consult your local IBM representative for information on the products and services currently available in your area.

ALTERNATIVE PRODUCTS AND SERVICES
Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

USE OF INFORMATION PROVIDED BY YOU
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

TRADEMARKS AND SERVICE MARKS
The following terms used in this publication are trademarks of International Business Machines Corporation in the United States, other countries or both:

- IBM
- WebSphere

Other company, product, and service names may be trademarks or service marks of others.

EXPORT REGULATIONS
You agree to comply with all applicable export and import laws and regulations.

Introduction:

The WebSphere Message Broker contains runtime transformation options which can be used to manipulate the data (logical message tree structure) passed through a message flow, in order to change an input message into an output message. A native installation of version 7 of the product provides:

- Compute node (containing ESQL transform code)
- JavaCompute node (containing Java transform code)
- XMLTransformation node (containing XSL transform code)
- PHP Compute node (containiing PHP code)
- Mapping node (drag and drop interface)

When embarking on a new WMB development, careful project consideration should be given to selecting which of these options is most suitable. Common factors to be taken into account include:

- Developer skill level with each technology
- Maintainability / reuse of the transformation code
- Performance of the transformation when deployed to the Broker runtime

For some projects, the utilisation of a graphical tool is a mandatory requirement in order to compensate for developers who have little experience with the other coding options, and also to ensure faster speeds of development for large and / or complex message models. In these circumstances, it can be a hard decision to select the graphical Mapping node option as it can mean sacrificing the advantages of code which is better optimised for performance and more easily maintained and reused.

Provided with this document, the MapToESQL Eclipse plugin for the Message Broker Toolkit, is designed to help in these situations.

The document provides step-by-step instructions for how to install and use the MapToESQL Eclipse plugin. It demonstrates its functional capability and using example message maps, describes the value the plugin can bring to Message Flow development activities.

Installation of the Plugin:

1. To install, make sure the WebSphere Message Broker Toolkit is closed.

2. Extract the supplied zip file named `ia9y_2.0.0.zip`

3. Copy the resulting subfolder, named `map2esql_2.0.0` into the eclipse installation's `plugins` subdirectory. If your broker was installed in the default location, then this directory will be located at:

   For WebSphere Message Broker version 7.0:

   ```
   C:\Program Files\IBM\WMBT700\plugins
   ```

   The version 1.0.0 release of this support pac which is now deprecated, was designed for use with WMBv6.0 and WMBv6.1 …

   For WebSphere Message Broker version 6.0:

   ```
   C:\Program Files\IBM\MessageBrokersToolkit\6.0\eclipse\plugins
   ```

   For WebSphere Message Broker version 6.1:

   ```
   C:\Program Files\IBM\WMBT610\plugins
   ```

4. Restart the WebSphere Message Broker Toolkit, using the "`-clean`" option:

   For WebSphere Message Broker version 7.0:

   ```
   "C:\Program Files\IBM\WMBT700\mb.exe" -clean
   ```

   The version 1.0.0 release of this support pac which is now deprecated, was designed for use with WMBv6.0 and WMBv6.1 …

   For WebSphere Message Broker version 6.0:

   ```
   C:\Program Files\IBM\MessageBrokersToolkit\6.0\wmbt.exe -clean
   ```

   For WebSphere Message Broker version 6.1:

   ```
   "C:\Program Files\IBM\WMBT610\eclipse.exe" -product
   com.ibm.etools.msgbroker.tooling.ide -clean
   ```

   For WebSphere Message Broker version 7.0:

   ```
   "C:\Program Files\IBM\WMBT700\mb.exe" -clean
   ```

Installation of the Samples:

The sample message sets and message flows have been updated to work with WMB version 7.0. If you attempt to use them with earlier versions, they may not work. The original samples which were produced for use with version 6.0 and version 6.1 are available in the project interchange file named ia9y_DeprecatedSampleProjects.zip. ***However, the instructions, screen shots and descriptions which follow in this document should be read in conjunction with the assets produced for version 7.0.***

1. From the WebSphere Message Broker Toolkit, select **`File`** → **`Import`**

2. From the Import wizard select the Import Source Project Interchange.

3. Click **`Next`**.

4. For the property **`From zip file`**, use the **`Browse`** button to navigate to the supplied zip file named **`ia9y_SampleProjects.zip`**

5. Select both the listed projects, **`MapToESQL_MessageFlows`** and **`MapToESQL_MessageSet`**.

6. Click **`Finish`**.

How to use the MapToESQL Plugin:

1. To invoke the Plugin, right-click a message map file in your Toolkit workspace, and from the context menu, select **`Map To ESQL`** → **`Generate ESQL`**.

2. The resulting **`Map To ESQL – Generate ESQL`** wizard asks you to specify whether or not you wish to use ESQL references, using checkboxes. An ESQL reference is a datatype which holds the location of a field in a message, analogous to the role of a pointer in most conventional programming languages. Using references in your ESQL code can lead to better performance, due to quicker navigation of the logical tree. Unfortunately, when dealing with complex transformations, references can also make ESQL harder to read. For this reason, the MapToESQL Plugin provides checkboxes so that you can specify what style of ESQL code should be produced.

3. The **`Map To ESQL – Generate ESQL`** wizard also contains dropdown menus, which should be used to specify which Input and Output Message Domains the generated code should use. The WebSphere Message Broker supplies a range of parsers to parse and write different message formats. Each parser is suited to a particular class of messages (for example, fixed-length binary, delimited text, or XML) known as a message domain. When a message flow input node, such as the WebSphere MQInput node for example, receives data from a queue it decides how the message should be parsed based upon the specified message domain. This domain can be provided as a node property or in the header of the message. Each message set that you create specifies a domain, which determines the parser that is used when parsing and writing messages that are defined within that message set. When you define a graphical message map, the message domain selected by the runtime is specified by examining the message set. However, when generating ESQL from a message map using the Plugin, it is necessary to inform the Plugin which message domain for input and output, you would like the generated code to use. The content of your map does not affect the message domain selections which are available for the generated ESQL code. The domains which you select will influence the syntax of the ESQL statements which refer to elements of the message body. If you would like, you can experiment with different domain settings and analyse the ESQL generated. When you have selected the Input and Output Message Domain, click Finish.

4. The Plugin will generate an ESQL file in the same project as the message map. The map will be named **`Generated_<Name of message map>.ESQL.`** If an ESQL file with this name already exists, it will be overwritten. The generated ESQL will contain a single Compute Module named:

   `<Name of message map>_Compute`

   The Module will contain a Function Main(), ready to be referenced from a Compute node.

5. To utilise the generated code in a message flow, replace the Mapping node with a Compute node. Right-click the Compute node in the palette and select **`Properties`**. For the **`ESQL Module`** property specify the Compute Module in the generated ESQL file:

   `<Name of message map>_Compute`

Example 1: Simple Mappings, MRM domain

Example 1 demonstrates the mapping of values from an input message parsed in the Message Repository Manager (MRM) domain to an output message also in the MRM domain. The values of fields in each of the Properties, MQMD and MQRFH2 folders are also copied from the input to the output.

Locate the project named **`MapToESQL_MessageFlows`**, and open the message map named **`MessageFlow1_Mapping`**. The map script is shown in Figure 1.

Map Script	Value
MessageFlow1_Mapping	
Parameters	
$target	
Properties	
MQMD	
MQRFH2C	
Version	$source/MQRFH2/Version
Format	$source/MQRFH2/Format
Encoding	$source/MQRFH2/Encoding
CodedCharSetId	$source/MQRFH2/CodedCharSetId
Flags	$source/MQRFH2/Flags
NameValueCCSID	$source/MQRFH2/NameValueCCSID
for	$source/MQRFH2/mcd
mcd	
Msd	$source/MQRFH2/mcd/Msd
Set	$source/MQRFH2/mcd/Set
Type	$source/MQRFH2/mcd/Type
Fmt	$source/MQRFH2/mcd/Fmt
message2	
localElementA	$source/message1/localElement1
localElementB	$source/message1/localElement2
localElementC	$source/message1/localElement3
localElementD	$source/message1/localElement4
localElementE	$source/message1/localElement5

Figure 1 MessageFlow1_Mapping

Note that this example also includes a simple “For” loop which iterates over the children of the mcd folder. More in-depth coverage of loop constructs is also provided with Example 2 and Example 3.

From the **`Broker Development`** View, right-click the Message Map named **`MessageFlow1_Mapping`** and from the context menu select **`Map To ESQL`** → **`Generate ESQL`**, as shown in Figure 2.

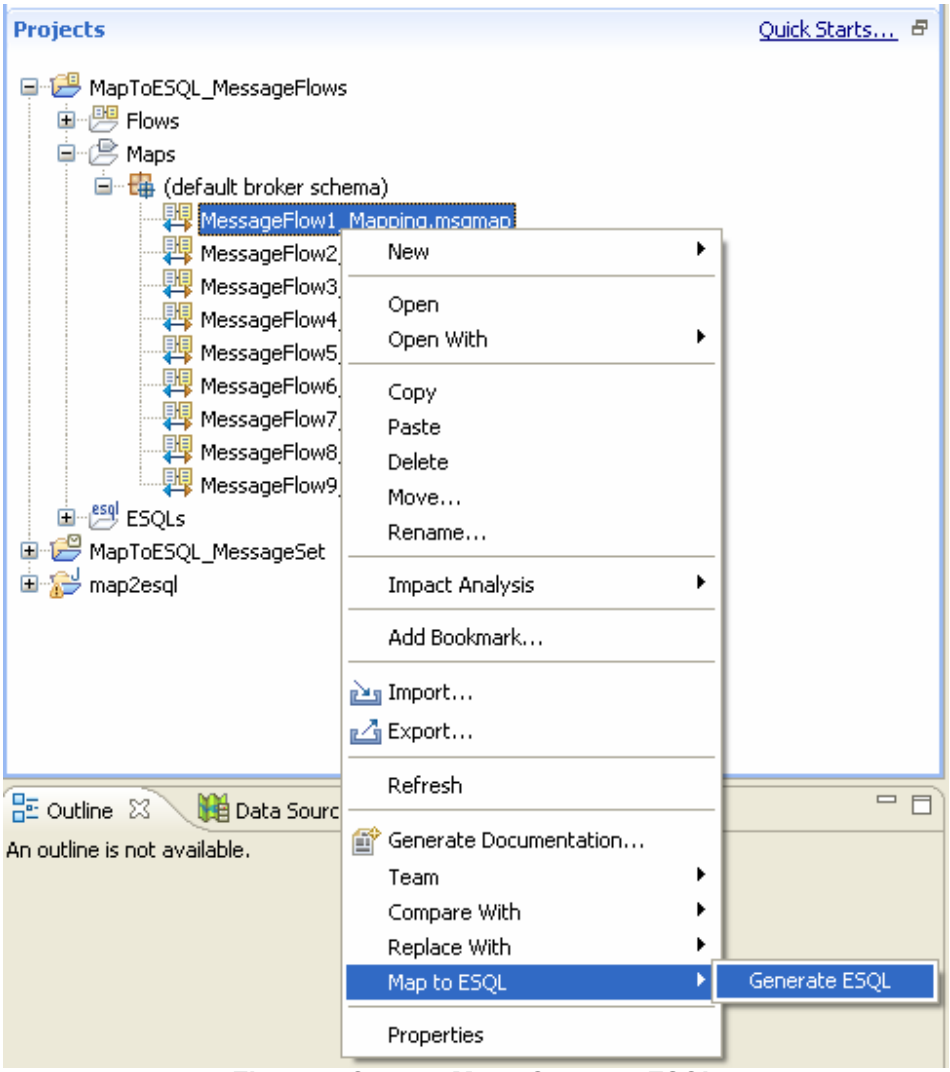


Figure 2 Context Menu Generate ESQL

This will start the plugin configuration wizard, as shown in Figure 3.

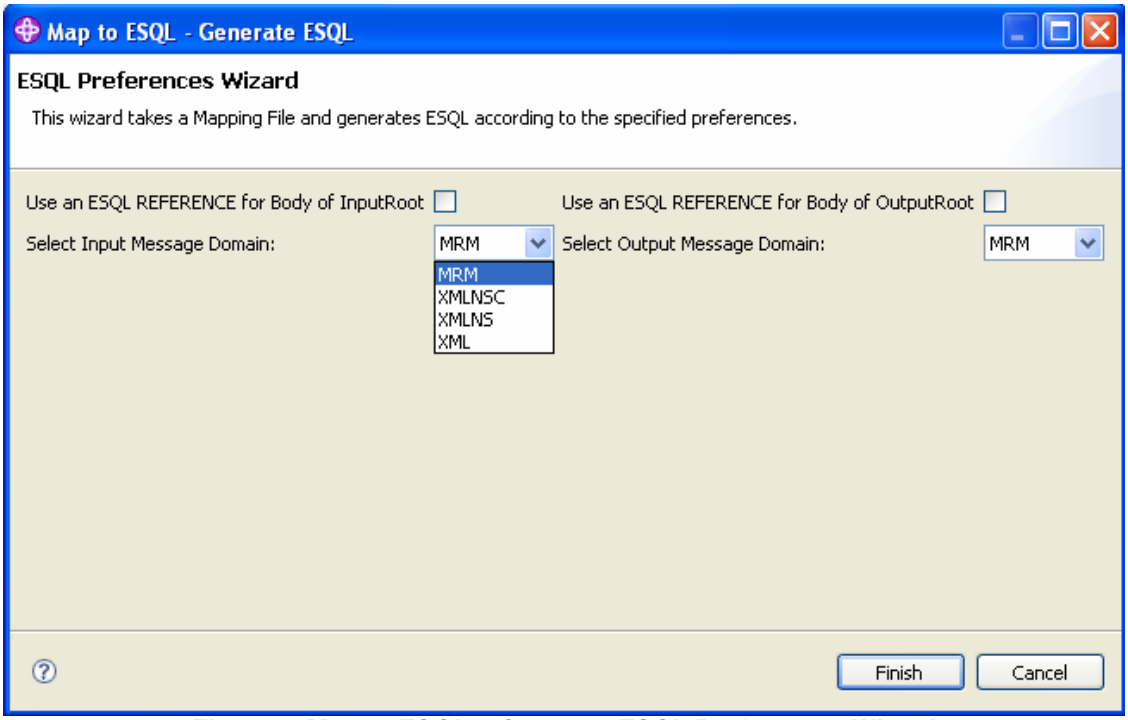


Figure 3 Map to ESQL – Generate ESQL Preferences Wizard

Figure 3 shows a screen shot of the wizard which is launched when you run the Plugin to Generate ESQL from a map. The Input message domain and output message domain are selected using dropdown menus. You can choose from the following message domains: **MRM, XMLNSC, XMLNS, XML**. To follow the documented Example 1, leave the default selection of **MRM** domain from both the dropdown menus and click the **Finish** button. Leave the check boxes unchecked. These features are explained in a later example. The message flow project will now contain a generated ESQL file named **Generated_MessageFlow1_Mapping.ESQL**

Generated_MessageFlow1_Mapping.esql

```
-- This file contains ESQL generated from a msgmap mapping file

CREATE COMPUTE MODULE MessageFlow1_Mapping_Compute

        CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN

            DECLARE xsd NAMESPACE 'http://www.w3.org/2001/XMLSchema';
            DECLARE xsi NAMESPACE 'http://www.w3.org/2001/XMLSchema-instance';
            SET OutputRoot.Properties.MessageSet = 'MapToESQL_MessageSet';
            SET OutputRoot.Properties.MessageType = '{}:message2';
            SET OutputRoot.Properties.MessageFormat = InputRoot.Properties.MessageFormat;
            SET OutputRoot.Properties.Encoding = InputRoot.Properties.Encoding;
            SET OutputRoot.Properties.CodedCharSetId = InputRoot.Properties.CodedCharSetId;
            SET OutputRoot.Properties.Transactional = InputRoot.Properties.Transactional;
            SET OutputRoot.Properties.Persistence = InputRoot.Properties.Persistence;
            SET OutputRoot.Properties.CreationTime = InputRoot.Properties.CreationTime;
            SET OutputRoot.Properties.ExpirationTime = InputRoot.Properties.ExpirationTime;
            SET OutputRoot.Properties.Priority = InputRoot.Properties.Priority;
            SET OutputRoot.Properties.ReplyIdentifier = InputRoot.Properties.ReplyIdentifier;
            SET OutputRoot.Properties.ReplyProtocol = InputRoot.Properties.ReplyProtocol;
            SET OutputRoot.Properties.Topic = InputRoot.Properties.Topic;
            SET OutputRoot.Properties.ContentType = InputRoot.Properties.ContentType;
            SET OutputRoot.Properties.IdentitySourceType = InputRoot.Properties.IdentitySourceType;
            SET OutputRoot.Properties.IdentitySourceToken = InputRoot.Properties.IdentitySourceToken;
            SET OutputRoot.Properties.IdentitySourcePassword = InputRoot.Properties.IdentitySourcePassword;
            SET OutputRoot.Properties.IdentitySourceIssuedBy = InputRoot.Properties.IdentitySourceIssuedBy;
            SET OutputRoot.Properties.IdentityMappedType = InputRoot.Properties.IdentityMappedType;
            SET OutputRoot.Properties.IdentityMappedToken = InputRoot.Properties.IdentityMappedToken;
            SET OutputRoot.Properties.IdentityMappedPassword = InputRoot.Properties.IdentityMappedPassword;
            SET OutputRoot.Properties.IdentityMappedIssuedBy = InputRoot.Properties.IdentityMappedIssuedBy;
            SET OutputRoot.MQMD.SourceQueue = InputRoot.MQMD.SourceQueue;
            SET OutputRoot.MQMD.Transactional = InputRoot.MQMD.Transactional;
            SET OutputRoot.MQMD.Encoding = InputRoot.MQMD.Encoding;
            SET OutputRoot.MQMD.CodedCharSetId = InputRoot.MQMD.CodedCharSetId;
            SET OutputRoot.MQMD.Format = InputRoot.MQMD.Format;
            SET OutputRoot.MQMD.Version = InputRoot.MQMD.Version;
```

Figure 4 Top Part of Generated ESQL From MessageFlow1_Mapping

```
            SET OutputRoot.MQRFH2C.Version = InputRoot.MQRFH2.Version;
            SET OutputRoot.MQRFH2C.Format = InputRoot.MQRFH2.Format;
            SET OutputRoot.MQRFH2C.Encoding = InputRoot.MQRFH2.Encoding;
            SET OutputRoot.MQRFH2C.CodedCharSetId = InputRoot.MQRFH2.CodedCharSetId;
            SET OutputRoot.MQRFH2C.Flags = InputRoot.MQRFH2.Flags;
            SET OutputRoot.MQRFH2C.NameValueCCSID = InputRoot.MQRFH2.NameValueCCSID;

            DECLARE A1 INTEGER 1;
            DECLARE A2 INTEGER CARDINALITY(InputRoot.MQRFH2.mcd[]);
            DECLARE A1_REF REFERENCE TO InputRoot.MQRFH2.mcd[1];
            WHILE A1 <= A2 DO
                MOVE A1_REF TO InputRoot.MQRFH2.mcd[A1];
                SET OutputRoot.MQRFH2C.mcd[A1].Msd = A1_REF.Msd;
                SET OutputRoot.MQRFH2C.mcd[A1].Set = A1_REF.Set;
                SET OutputRoot.MQRFH2C.mcd[A1].Type = A1_REF.Type;
                SET OutputRoot.MQRFH2C.mcd[A1].Fmt = A1_REF.Fmt;
                SET A1 = A1 + 1;
            END WHILE;

            SET OutputRoot.MRM.localElementA = InputRoot.MRM.localElement1;
            SET OutputRoot.MRM.localElementB = InputRoot.MRM.localElement2;
            SET OutputRoot.MRM.localElementC = InputRoot.MRM.localElement3;
            SET OutputRoot.MRM.localElementD = InputRoot.MRM.localElement4;
            SET OutputRoot.MRM.localElementE = InputRoot.MRM.localElement5;

        RETURN TRUE;
        END;

END MODULE;
```

Figure 5 Bottom Part of Generated ESQL From MessageFlow1_Mapping

Example 2: FOR loops, MRM and XMLNSC domains

Example 2 demonstrates the mapping of values from an input message parsed in the XMLNSC domain to an output message in the MRM domain. The values of fields in the Properties Folder and MQMD header are also copied from the input to the output.

Locate the project named `MapToESQL_MessageFlows`, and open the message map named `MessageFlow2_Mapping`. The map script is shown in Figure 7.

Map Script	Value
MessageFlow2_Mapping	
Parameters	
$target	
Properties	
MQMD	$source/MQMD
message4	
localElement2	
for	$source/message3/localElement1/localElementA
localElementD	$source/message3/localElement1/localElementA
for	$source/message3/localElement1/localElementB
localElementE	$source/message3/localElement1/localElementB
for	$source/message3/localElement1/localElementC
localElementF	$source/message3/localElement1/localElementC

Figure 6 MessageFlow2_Mapping

The input message includes three localElements named `localElementA`, `localElementB` and `localElementC`, which can occur between 1 and 5 times. Likewise the output message contains three localElements which can also repeat. This use case demonstrates the way that For loops contained in message maps are translated into For loops in the ESQL which is generated. The generated ESQL code uses Integer variables as loop indices. The plugin successfully handles maps which contain more than one loop, and also deals with nested elements which repeat.

Figure 7 shows a screen shot of the wizard which is launched when you run the Plugin to Generate ESQL from a map. The Input message domain and output message domain are selected using dropdown menus. You can choose from the following message domains: `MRM`, `XMLNSC`, `XMLNS`, `XML`. To follow the documented Example 2, select an `Input Message Domain` of `XMLNSC` and leave the `Output Message Domain` as the default selection of `MRM`. Click the `Finish` button.

The message flow project will now contain a generated ESQL file named `Generated_MessageFlow2_Mapping.ESQL`

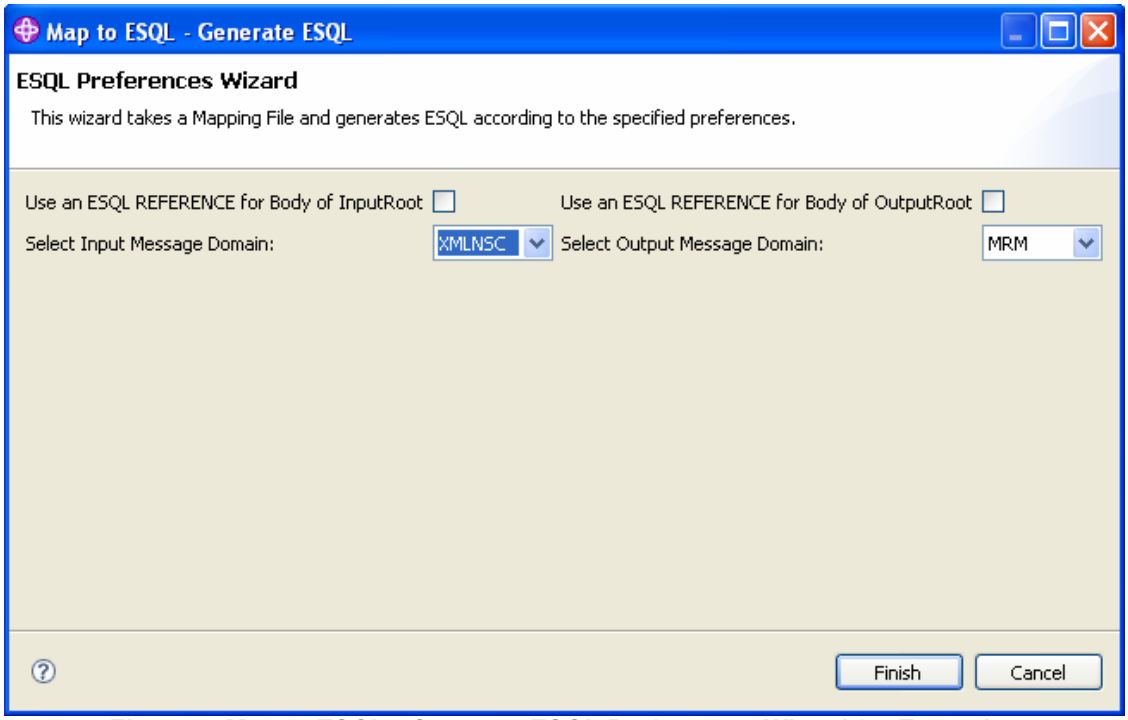


Figure 7 Map to ESQL - Generate ESQL Preferences Wizard for Example2

Generated_MessageFlow2_Mapping.esql

```
-- This file contains ESQL generated from a msgmap mapping file

CREATE COMPUTE MODULE MessageFlow2_Mapping_Compute

        CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN

            DECLARE xsd NAMESPACE 'http://www.w3.org/2001/XMLSchema';
            SET OutputRoot.Properties.MessageSet = 'L51JS40002001';
            SET OutputRoot.Properties.MessageType = 'message4';
            SET OutputRoot.Properties.MessageFormat = InputRoot.Properties.MessageFormat;
            SET OutputRoot.Properties.Encoding = InputRoot.Properties.Encoding;
            SET OutputRoot.Properties.CodedCharSetId = InputRoot.Properties.CodedCharSetId;
            SET OutputRoot.Properties.Transactional = InputRoot.Properties.Transactional;
            SET OutputRoot.Properties.Persistence = InputRoot.Properties.Persistence;
            SET OutputRoot.Properties.CreationTime = InputRoot.Properties.CreationTime;
            SET OutputRoot.Properties.ExpirationTime = InputRoot.Properties.ExpirationTime;
            SET OutputRoot.Properties.Priority = InputRoot.Properties.Priority;
            SET OutputRoot.Properties.ReplyIdentifier = InputRoot.Properties.ReplyIdentifier;
            SET OutputRoot.Properties.ReplyProtocol = InputRoot.Properties.ReplyProtocol;
            SET OutputRoot.Properties.Topic = InputRoot.Properties.Topic;
            SET OutputRoot.Properties.ContentType = InputRoot.Properties.ContentType;
            SET OutputRoot.MQMD = InputRoot.MQMD;

            DECLARE A1 INTEGER 1;
            DECLARE A2 INTEGER CARDINALITY(InputRoot.XMLNSC.message3.localElement1.localElementA[]);
            DECLARE A1_REF REFERENCE TO InputRoot.XMLNSC.message3.localElement1.localElementA[1];
            WHILE A1 <= A2 DO
                MOVE A1_REF TO InputRoot.XMLNSC.message3.localElement1.localElementA[A1];
                SET OutputRoot.MRM.localElement2.localElementD[A1] = A1_REF;
                SET A1 = A1 + 1;
            END WHILE;

            DECLARE A3 INTEGER 1;
            DECLARE A4 INTEGER CARDINALITY(InputRoot.XMLNSC.message3.localElement1.localElementB[]);
            DECLARE A3_REF REFERENCE TO InputRoot.XMLNSC.message3.localElement1.localElementB[1];
            WHILE A3 <= A4 DO
                MOVE A3_REF TO InputRoot.XMLNSC.message3.localElement1.localElementB[A3];
                SET OutputRoot.MRM.localElement2.localElementE[A3] = A3_REF;
                SET A3 = A3 + 1;
            END WHILE;


            DECLARE A5 INTEGER 1;
            DECLARE A6 INTEGER CARDINALITY(InputRoot.XMLNSC.message3.localElement1.localElementC[]);
            DECLARE A5_REF REFERENCE TO InputRoot.XMLNSC.message3.localElement1.localElementC[1];
            WHILE A5 <= A6 DO
                MOVE A5_REF TO InputRoot.XMLNSC.message3.localElement1.localElementC[A5];
                SET OutputRoot.MRM.localElement2.localElementF[A5] = A5_REF;
                SET A5 = A5 + 1;
            END WHILE;


        RETURN TRUE;
        END;

END MODULE;
```

Figure 8 Generated ESQL from MessageFlow2_Mapping

Example 3: Nested FOR loops, MRM domain

Example 3 demonstrates the mapping of values from an input message parsed in the MRM domain to an output message also in the MRM domain. The values of fields in the Properties Folder and MQMD header are copied from the input to the output. The message body itself includes three fields which repeat, whose mapping is achieved using For loops. This example is more complex than Example 2 as it includes nested loops and a mixture of repeating and non-repeating items.

Locate the project named `MapToESQL_MessageFlows`, and open the message map named `MessageFlow3_Mapping`. The map script is shown in Figure 9.

Map Script	Value
MessageFlow3_Mapping	
Parameters	
$target	
Properties	
MessageSet	"MapToESQL_MessageSet"
MessageType	"{}:message6"
MessageFormat	$source/Properties/MessageFormat
Encoding	$source/Properties/Encoding
CodedCharSetId	$source/Properties/CodedCharSetId
Transactional	$source/Properties/Transactional
Persistence	$source/Properties/Persistence
CreationTime	$source/Properties/CreationTime
ExpirationTime	$source/Properties/ExpirationTime
Priority	$source/Properties/Priority
ReplyIdentifier	$source/Properties/ReplyIdentifier
ReplyProtocol	$source/Properties/ReplyProtocol
Topic	$source/Properties/Topic
ContentType	$source/Properties/ContentType
IdentitySourceType	$source/Properties/IdentitySourceType
IdentitySourceToken	$source/Properties/IdentitySourceToken
IdentitySourcePassword	$source/Properties/IdentitySourcePassword
IdentitySourceIssuedBy	$source/Properties/IdentitySourceIssuedBy
IdentityMappedType	$source/Properties/IdentityMappedType
IdentityMappedToken	$source/Properties/IdentityMappedToken
IdentityMappedPassword	$source/Properties/IdentityMappedPassword
IdentityMappedIssuedBy	$source/Properties/IdentityMappedIssuedBy
MQMD	$source/MQMD
message6	
for	$source/message5/level1Element
mappedlevel1Element	
mappednonRepeat1A	$source/message5/level1Element/nonRepeat1A
for	$source/message5/level1Element/level2Element
mappedlevel2Element	
mappednonRepeat2	$source/message5/level1Element/level2Element/nonRepeat2
for	$source/message5/level1Element/level2Element/level3Element
mappedlevel3Element	$source/message5/level1Element/level2Element/level3Element
mappednonRepeat1B	$source/message5/level1Element/nonRepeat1B

Figure 9 MessageFlow3_Mapping

Figure 9 shows the map script from which the ESQL is generated. Note that the output message includes localElements at three levels. `mappedLevel1Element`, has a child `mappedlevel2Element` which in turn has a child `mappedLevel3Element`. `mappedLevel1Element` also has two children which do not repeat and are defined either side of the second for loop. This complex structure is catered for by the plugin. Figure 10 shows the ESQL which is produced when the plugin is run with an input and output domain set to MRM. In

particular your attention is drawn to the loop indices, named A, C and E. Note also the insertion of the SET statements within the correct loops.

Generated_MessageFlow3_Mapping.esql

```
-- This file contains ESQL generated from a msgmap mapping file

CREATE COMPUTE MODULE MessageFlow3_Mapping_Compute

        CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN

            DECLARE xsd NAMESPACE 'http://www.w3.org/2001/XMLSchema';
            SET OutputRoot.Properties.MessageSet = 'L51JS40002001';
            SET OutputRoot.Properties.MessageType = 'message6';
            SET OutputRoot.Properties.MessageFormat = InputRoot.Properties.MessageFormat;
            SET OutputRoot.Properties.Encoding = InputRoot.Properties.Encoding;
            SET OutputRoot.Properties.CodedCharSetId = InputRoot.Properties.CodedCharSetId;
            SET OutputRoot.Properties.Transactional = InputRoot.Properties.Transactional;
            SET OutputRoot.Properties.Persistence = InputRoot.Properties.Persistence;
            SET OutputRoot.Properties.CreationTime = InputRoot.Properties.CreationTime;
            SET OutputRoot.Properties.ExpirationTime = InputRoot.Properties.ExpirationTime;
            SET OutputRoot.Properties.Priority = InputRoot.Properties.Priority;
            SET OutputRoot.Properties.ReplyIdentifier = InputRoot.Properties.ReplyIdentifier;
            SET OutputRoot.Properties.ReplyProtocol = InputRoot.Properties.ReplyProtocol;
            SET OutputRoot.Properties.Topic = InputRoot.Properties.Topic;
            SET OutputRoot.Properties.ContentType = InputRoot.Properties.ContentType;
            SET OutputRoot.MQMD = InputRoot.MQMD;

            DECLARE A1 INTEGER 1;
            DECLARE A2 INTEGER CARDINALITY(InputRoot.MRM.level1Element[]);
            DECLARE A1_REF REFERENCE TO InputRoot.MRM.level1Element[1];
            WHILE A1 <= A2 DO
                MOVE A1_REF TO InputRoot.MRM.level1Element[A1];
                SET OutputRoot.MRM.mappedlevel1Element[A1].mappednonRepeat1A = A1_REF.nonRepeat1A;

                DECLARE A3 INTEGER 1;
                DECLARE A4 INTEGER CARDINALITY(InputRoot.MRM.level1Element[A1].level2Element[]);
                DECLARE A3_REF REFERENCE TO InputRoot.MRM.level1Element[A1].level2Element[1];
                WHILE A3 <= A4 DO
                    MOVE A3_REF TO InputRoot.MRM.level1Element[A1].level2Element[A3];
                    SET OutputRoot.MRM.mappedlevel1Element[A1].mappedlevel2Element[A3].mappednonRepeat2 = A3_REF.nonRepeat2;

                    DECLARE A5 INTEGER 1;
                    DECLARE A6 INTEGER CARDINALITY(InputRoot.MRM.level1Element[A1].level2Element[A3].level3Element[]);
                    DECLARE A5_REF REFERENCE TO InputRoot.MRM.level1Element[A1].level2Element[A3].level3Element[1];
                    WHILE A5 <= A6 DO
                        MOVE A5_REF TO InputRoot.MRM.level1Element[A1].level2Element[A3].level3Element[A5];
                        SET OutputRoot.MRM.mappedlevel1Element[A1].mappedlevel2Element[A3].mappedlevel3Element[A5] = A5_REF;
                        SET A5 = A5 + 1;
                    END WHILE;

                    SET A3 = A3 + 1;
                END WHILE;
                SET OutputRoot.MRM.mappedlevel1Element[A1].mappednonRepeat1B = A1_REF.nonRepeat1B;
                SET A1 = A1 + 1;
            END WHILE;

        RETURN TRUE;
        END;

END MODULE;
```

Figure 10 Generated ESQL from MessageFlow3_Mapping

Example 4: XPath STRING Functions, MRM domain

Example 4 demonstrates the mapping of values from an input message parsed in the MRM domain to an output message also in the MRM domain. The values of fields in the Properties Folder and MQMD header are copied from the input to the output. The message body itself includes two fields whose output values are generated using XPath String functions.

Locate the project named `MapToESQL_MessageFlows`, and open the message map named `MessageFlow4_Mapping`. The map script is shown in Figure 11.

Map Script	Value
MessageFlow4_Mapping	
Parameters	
$target	
Properties	
MessageSet	"MapToESQL_MessageSet"
MessageType	"{}:message8"
MessageFormat	$source/Properties/MessageFormat
Encoding	$source/Properties/Encoding
CodedCharSetId	$source/Properties/CodedCharSetId
Transactional	$source/Properties/Transactional
Persistence	$source/Properties/Persistence
CreationTime	$source/Properties/CreationTime
ExpirationTime	$source/Properties/ExpirationTime
Priority	$source/Properties/Priority
ReplyIdentifier	$source/Properties/ReplyIdentifier
ReplyProtocol	$source/Properties/ReplyProtocol
Topic	$source/Properties/Topic
ContentType	$source/Properties/ContentType
IdentitySourceType	$source/Properties/IdentitySourceType
IdentitySourceToken	$source/Properties/IdentitySourceToken
IdentitySourcePassword	$source/Properties/IdentitySourcePassword
IdentitySourceIssuedBy	$source/Properties/IdentitySourceIssuedBy
IdentityMappedType	$source/Properties/IdentityMappedType
IdentityMappedToken	$source/Properties/IdentityMappedToken
IdentityMappedPassword	$source/Properties/IdentityMappedPassword
IdentityMappedIssuedBy	$source/Properties/IdentityMappedIssuedBy
MQMD	$source/MQMD
message8	
localElement1	fn:concat($source/message7/localElement1,$source/message7/localElement2)
localElement2	fn:substring($source/message7/localElement3,1,3)

Figure 11 MessageFlow4_Mapping

Figure 11 shows the map script from which the ESQL is generated. Note that the output message includes `localElement1` which takes its output value from the concatenation of two string values in the input message. The output message also contains `localElement2` which takes its value from substringing the first three characters of the input message's `localElement3`.

The XPath `concat` function supports the concatenation of multiple (more than two) string values.

The XPath `substring` function expects three arguments: The string from which the substring is extracted, the starting position from which the substring runs, and the number of characters to be taken. If the last argument is omitted, then the substring will run to the very end of the first argument's data.

Generated_MessageFlow4_Mapping.esql

```
-- This file contains ESQL generated from a msgmap mapping file

CREATE COMPUTE MODULE MessageFlow4_Mapping_Compute

        CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN

            DECLARE xsd NAMESPACE 'http://www.w3.org/2001/XMLSchema';
            SET OutputRoot.Properties.MessageSet = 'L51JS40002001';
            SET OutputRoot.Properties.MessageType = 'message8';
            SET OutputRoot.Properties.MessageFormat = InputRoot.Properties.MessageFormat;
            SET OutputRoot.Properties.Encoding = InputRoot.Properties.Encoding;
            SET OutputRoot.Properties.CodedCharSetId = InputRoot.Properties.CodedCharSetId;
            SET OutputRoot.Properties.Transactional = InputRoot.Properties.Transactional;
            SET OutputRoot.Properties.Persistence = InputRoot.Properties.Persistence;
            SET OutputRoot.Properties.CreationTime = InputRoot.Properties.CreationTime;
            SET OutputRoot.Properties.ExpirationTime = InputRoot.Properties.ExpirationTime;
            SET OutputRoot.Properties.Priority = InputRoot.Properties.Priority;
            SET OutputRoot.Properties.ReplyIdentifier = InputRoot.Properties.ReplyIdentifier;
            SET OutputRoot.Properties.ReplyProtocol = InputRoot.Properties.ReplyProtocol;
            SET OutputRoot.Properties.Topic = InputRoot.Properties.Topic;
            SET OutputRoot.Properties.ContentType = InputRoot.Properties.ContentType;
            SET OutputRoot.MQMD = InputRoot.MQMD;
            SET OutputRoot.MRM.localElement1 = InputRoot.MRM.localElement1 || InputRoot.MRM.localElement2;
            SET OutputRoot.MRM.localElement2 = SUBSTRING(InputRoot.MRM.localElement3 FROM 1 FOR 3);

        RETURN TRUE;
        END;

END MODULE;
```

Figure 12 Generated ESQL from MessageFlow4_Mapping

Example 5: User Defined ESQL Functions, MRM domain

Example 5 demonstrates the mapping of values from an input message parsed in the MRM domain to an output message also in the MRM domain. The values of fields in the Properties Folder and MQMD header are copied from the input to the output. The message body itself includes two fields whose output values are generated using ESQL functions. Locate the project named `MapToESQL_MessageFlows`, and open the message map named `MessageFlow5_Mapping`. The map script is shown in Figure 13.

Map Script	Value
MessageFlow5_Mapping	
Parameters	
$target	
Properties	
MessageSet	"MapToESQL_MessageSet"
MessageType	"{}:message10"
MessageFormat	$source/Properties/MessageFormat
Encoding	$source/Properties/Encoding
CodedCharSetId	$source/Properties/CodedCharSetId
Transactional	$source/Properties/Transactional
Persistence	$source/Properties/Persistence
CreationTime	$source/Properties/CreationTime
ExpirationTime	$source/Properties/ExpirationTime
Priority	$source/Properties/Priority
ReplyIdentifier	$source/Properties/ReplyIdentifier
ReplyProtocol	$source/Properties/ReplyProtocol
Topic	$source/Properties/Topic
ContentType	$source/Properties/ContentType
IdentitySourceType	$source/Properties/IdentitySourceType
IdentitySourceToken	$source/Properties/IdentitySourceToken
IdentitySourcePassword	$source/Properties/IdentitySourcePassword
IdentitySourceIssuedBy	$source/Properties/IdentitySourceIssuedBy
IdentityMappedType	$source/Properties/IdentityMappedType
IdentityMappedToken	$source/Properties/IdentityMappedToken
IdentityMappedPassword	$source/Properties/IdentityMappedPassword
IdentityMappedIssuedBy	$source/Properties/IdentityMappedIssuedBy
MQMD	$source/MQMD
message10	
resultElement1	esql:Cardinality($source/message9/localElement1)
resultElement2	esql:Average($source/message9/localElement2,$source/message9/localElement3)

Figure 13 MessageFlow5_Mapping

Figure 13 shows the map script from which the ESQL is generated. The output message includes a `resultElement1` which takes its output value from the ESQL function named `Cardinality` and a `resultElement2` which takes its output value from the ESQL function named `Average`. These two ESQL functions are defined in the file named `Library.esql` which you will find in the same message flow project, named `MapToESQL_MessageFlows`.

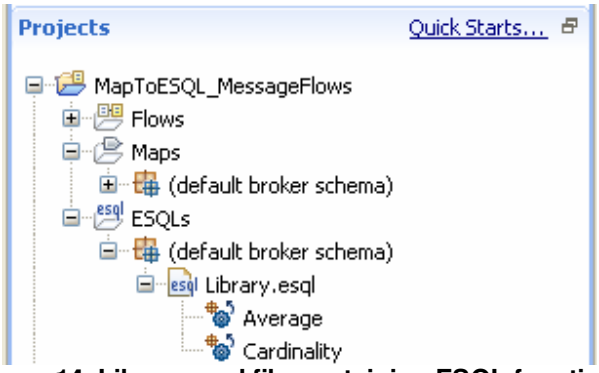


Figure 14 Library.esql file containing ESQL functions

```
CREATE FUNCTION Average(IN P1 CHAR, IN P2 CHAR) RETURNS INTEGER
BEGIN
    DECLARE I INTEGER ((CAST(P1 AS INTEGER) + CAST(P2 AS INTEGER)) / 2 );
    RETURN I;
END;

CREATE FUNCTION Cardinality(IN source REFERENCE) RETURNS INTEGER
BEGIN
    DECLARE I INTEGER CARDINALITY(source.*[]);
    RETURN I;
END;
```

Figure 15 ESQL Average and Cardinality Functions

Figure 15 shows the contents of `Library.esql` which contains the two ESQL functions, `Average` and `Cardinality`. Figure 16 shows the generated ESQL:

```
Generated_MessageFlow5_Mapping.esql
-- This file contains ESQL generated from a msgmap mapping file

CREATE COMPUTE MODULE MessageFlow5_Mapping_Compute

        CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN

            DECLARE xsd NAMESPACE 'http://www.w3.org/2001/XMLSchema';
            SET OutputRoot.Properties.MessageSet = 'L51JS40002001';
            SET OutputRoot.Properties.MessageType = 'message10';
            SET OutputRoot.Properties.MessageFormat = InputRoot.Properties.MessageFormat;
            SET OutputRoot.Properties.Encoding = InputRoot.Properties.Encoding;
            SET OutputRoot.Properties.CodedCharSetId = InputRoot.Properties.CodedCharSetId;
            SET OutputRoot.Properties.Transactional = InputRoot.Properties.Transactional;
            SET OutputRoot.Properties.Persistence = InputRoot.Properties.Persistence;
            SET OutputRoot.Properties.CreationTime = InputRoot.Properties.CreationTime;
            SET OutputRoot.Properties.ExpirationTime = InputRoot.Properties.ExpirationTime;
            SET OutputRoot.Properties.Priority = InputRoot.Properties.Priority;
            SET OutputRoot.Properties.ReplyIdentifier = InputRoot.Properties.ReplyIdentifier;
            SET OutputRoot.Properties.ReplyProtocol = InputRoot.Properties.ReplyProtocol;
            SET OutputRoot.Properties.Topic = InputRoot.Properties.Topic;
            SET OutputRoot.Properties.ContentType = InputRoot.Properties.ContentType;
            SET OutputRoot.MQMD = InputRoot.MQMD;
            SET OutputRoot.MRM.resultElement1 = Cardinality(InputRoot.MRM.localElement1);
            SET OutputRoot.MRM.resultElement2 = Average(InputRoot.MRM.localElement2,InputRoot.MRM.localElement3);

        RETURN TRUE;
        END;

END MODULE;
```

Figure 16 Generated ESQL from MessageFlow5_Mapping

Example 6: IF and ELSE clauses, MRM domain

Example 6 demonstrates the mapping of values from an input message parsed in the MRM domain to an output message also in the MRM domain. The values of fields in the Properties Folder and MQMD header are copied from the input to the output. The message body itself includes the string field named `localElement1` whose output value is dependent on the input `localElement1` of type `xsd:int`.

Locate the project named `MapToESQL_MessageFlows`, and open the message map named `MessageFlow6_Mapping`. The map script is shown in Figure 17.

Map Script	Value
MessageFlow6_Mapping	
Parameters	
$target	
Properties	
MQMD	$source/MQMD
message12	
if	$source/message11/localElement1 < 5
localElement1	"The input localElement1 contained a value smaller than 5!"
elseif	$source/message11/localElement1 > 5
localElement1	"The input localElement1 contained a value bigger than 5!"
else	
localElement1	"The input localElement1 contained the value 5!"

Figure 17 MessageFlow6_Mapping

Figure 17 shows the map script from which the ESQL is generated. The output message contains a single localElement named `localElement1`. The string value which is assigned to this element depends upon the `xsd:int` value of the input message's `localElement1`. The `if` `elseif` and `else` clauses define three possible outcomes, dependent on whether the integer value is less than, greater than or equal to the value 5.

Figure 18 shows the generated ESQL.

Generated_MessageFlow6_Mapping.esql

```
-- This file contains ESQL generated from a msgmap mapping file

CREATE COMPUTE MODULE MessageFlow6_Mapping_Compute

        CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN

            DECLARE xsd NAMESPACE 'http://www.w3.org/2001/XMLSchema';
            SET OutputRoot.Properties.MessageSet = 'L51JS40002001';
            SET OutputRoot.Properties.MessageType = 'message12';
            SET OutputRoot.Properties.MessageFormat = InputRoot.Properties.MessageFormat;
            SET OutputRoot.Properties.Encoding = InputRoot.Properties.Encoding;
            SET OutputRoot.Properties.CodedCharSetId = InputRoot.Properties.CodedCharSetId;
            SET OutputRoot.Properties.Transactional = InputRoot.Properties.Transactional;
            SET OutputRoot.Properties.Persistence = InputRoot.Properties.Persistence;
            SET OutputRoot.Properties.CreationTime = InputRoot.Properties.CreationTime;
            SET OutputRoot.Properties.ExpirationTime = InputRoot.Properties.ExpirationTime;
            SET OutputRoot.Properties.Priority = InputRoot.Properties.Priority;
            SET OutputRoot.Properties.ReplyIdentifier = InputRoot.Properties.ReplyIdentifier;
            SET OutputRoot.Properties.ReplyProtocol = InputRoot.Properties.ReplyProtocol;
            SET OutputRoot.Properties.Topic = InputRoot.Properties.Topic;
            SET OutputRoot.Properties.ContentType = InputRoot.Properties.ContentType;
            SET OutputRoot.MQMD = InputRoot.MQMD;
            IF InputRoot.MRM.localElement1 < 5 THEN
                SET OutputRoot.MRM.localElement1 = 'The input localElement1 contained a value smaller than 5!';
            ELSEIF InputRoot.MRM.localElement1 > 5 THEN
                SET OutputRoot.MRM.localElement1 = 'The input localElement1 contained a value bigger than 5!';
            ELSE
                SET OutputRoot.MRM.localElement1 = 'The input localElement1 contained a value smaller than 5!';
            END IF;

        RETURN TRUE;
        END;

END MODULE;
```

Figure 18 Generated ESQL from MessageFlow6_Mapping

Example 7: Multiple Header Support, MRM domain

Example 7 demonstrates the mapping of values from an input message parsed in the MRM domain to an output message also in the MRM domain. The values of all fields in all supported headers are copied from the input to the output. The mapping of the message body itself includes copying values from five input fields to five output fields (differently named).

Locate the project named `MapToESQL_MessageFlows`, and open the message map named `MessageFlow7_Mapping`. The mapping itself is shown in Figure 19, and the Map Script is shown in Figure 20.

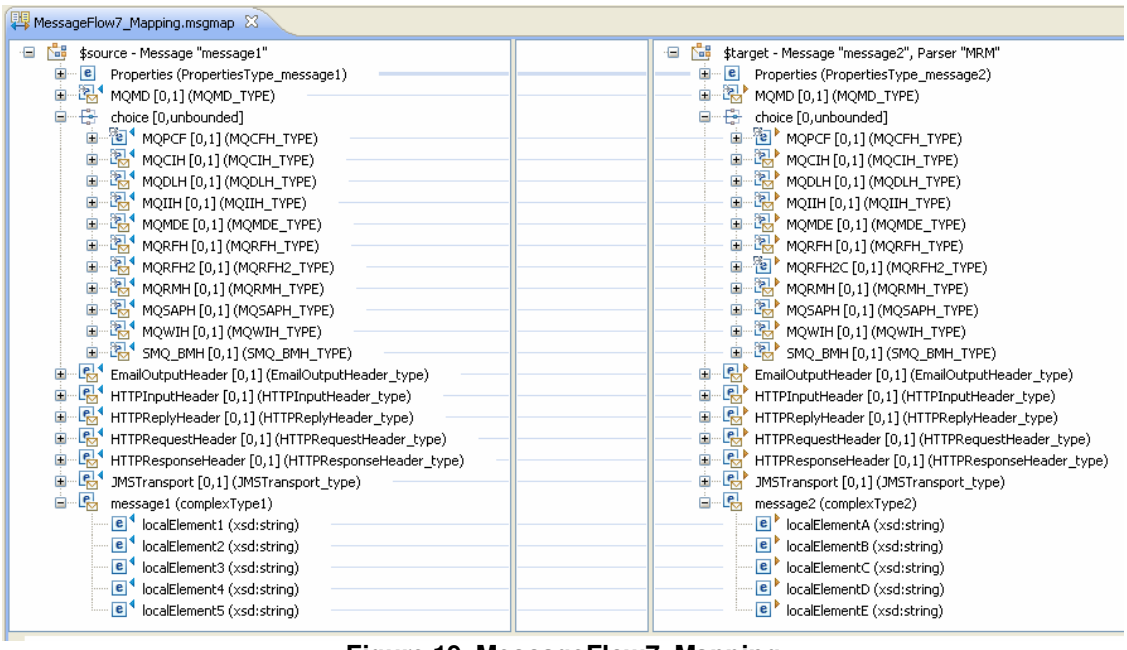


Figure 19 MessageFlow7_Mapping

Map Script	Value
MessageFlow7_Mapping	
Parameters	
$target	
Properties	
for	$source/MQMD
for	$source/MQPCF
for	$source/MQCIH
for	$source/MQDLH
for	$source/MQIIH
for	$source/MQMDE
for	$source/MQRFH
for	$source/MQRFH2
for	$source/MQRMH
for	$source/MQSAPH
for	$source/MQWIH
for	$source/SMQ_BMH
for	$source/EmailOutputHeader
for	$source/HTTPInputHeader
for	$source/HTTPReplyHeader
for	$source/HTTPRequestHeader
for	$source/HTTPResponseHeader
for	$source/JMSTransport
message2	
localElementA	$source/message1/localElement1
localElementB	$source/message1/localElement2
localElementC	$source/message1/localElement3
localElementD	$source/message1/localElement4
localElementE	$source/message1/localElement5

Figure 20 MessageFlow7_Mapping Map Script

The ESQL which is generated from the Mapping, contains a series of For loops, one for each header. The For loops ensure that the header is created on output if it was present in the input. The reason for this sample is to demonstrate the header support within the Plugin. In real life, it is highly unlikely that a logical message tree would contain all of the headers shown. The first part of the generated ESQL is shown in Figure 21:

Generated_MessageFlow7_Mapping.esql

```
-- This file contains ESQL generated from a msgmap mapping file

CREATE COMPUTE MODULE MessageFlow7_Mapping_Compute

        CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN

            DECLARE xsd NAMESPACE 'http://www.w3.org/2001/XMLSchema';
            DECLARE xsi NAMESPACE 'http://www.w3.org/2001/XMLSchema-instance';
            SET OutputRoot.Properties.MessageSet = 'MapToESQL_MessageSet';
            SET OutputRoot.Properties.MessageType = '{}:message2';
            SET OutputRoot.Properties.MessageFormat = InputRoot.Properties.MessageFormat;
            SET OutputRoot.Properties.Encoding = InputRoot.Properties.Encoding;
            SET OutputRoot.Properties.CodedCharSetId = InputRoot.Properties.CodedCharSetId;
            SET OutputRoot.Properties.Transactional = InputRoot.Properties.Transactional;
            SET OutputRoot.Properties.Persistence = InputRoot.Properties.Persistence;
            SET OutputRoot.Properties.CreationTime = InputRoot.Properties.CreationTime;
            SET OutputRoot.Properties.ExpirationTime = InputRoot.Properties.ExpirationTime;
            SET OutputRoot.Properties.Priority = InputRoot.Properties.Priority;
            SET OutputRoot.Properties.ReplyIdentifier = InputRoot.Properties.ReplyIdentifier;
            SET OutputRoot.Properties.ReplyProtocol = InputRoot.Properties.ReplyProtocol;
            SET OutputRoot.Properties.Topic = InputRoot.Properties.Topic;
            SET OutputRoot.Properties.ContentType = InputRoot.Properties.ContentType;
            SET OutputRoot.Properties.IdentitySourceType = InputRoot.Properties.IdentitySourceType;
            SET OutputRoot.Properties.IdentitySourceToken = InputRoot.Properties.IdentitySourceToken;
            SET OutputRoot.Properties.IdentitySourcePassword = InputRoot.Properties.IdentitySourcePassword;
            SET OutputRoot.Properties.IdentitySourceIssuedBy = InputRoot.Properties.IdentitySourceIssuedBy;
            SET OutputRoot.Properties.IdentityMappedType = InputRoot.Properties.IdentityMappedType;
            SET OutputRoot.Properties.IdentityMappedToken = InputRoot.Properties.IdentityMappedToken;
            SET OutputRoot.Properties.IdentityMappedPassword = InputRoot.Properties.IdentityMappedPassword;
            SET OutputRoot.Properties.IdentityMappedIssuedBy = InputRoot.Properties.IdentityMappedIssuedBy;

            DECLARE A1 INTEGER 1;
            DECLARE A2 INTEGER CARDINALITY(InputRoot.MQMD[]);
            DECLARE A1_REF REFERENCE TO InputRoot.MQMD[1];
            WHILE A1 <= A2 DO
                MOVE A1_REF TO InputRoot.MQMD[A1];
                SET OutputRoot.MQMD[A1] = A1_REF;
                SET A1 = A1 + 1;
            END WHILE;
```

Figure 21 First section of Generated ESQL from MessageFlow7_Mapping

Example 8: Namespace Support, MRM domain

Example 8 demonstrates the mapping of values from an input message parsed in the MRM domain to an output message also in the MRM domain. The input message's elements and the output message's elements exist in Target Namespaces (not in the default NoTargetNamespace). This example demonstrates that the ESQL generated by the Plugin is sensitive to the required namespace prefixes.

Locate the project named `MapToESQL_MessageFlows`, and open the message map named `MessageFlow8_Mapping`. The mapping itself is shown in Figure 22, and the Map Script is shown in Figure 23.

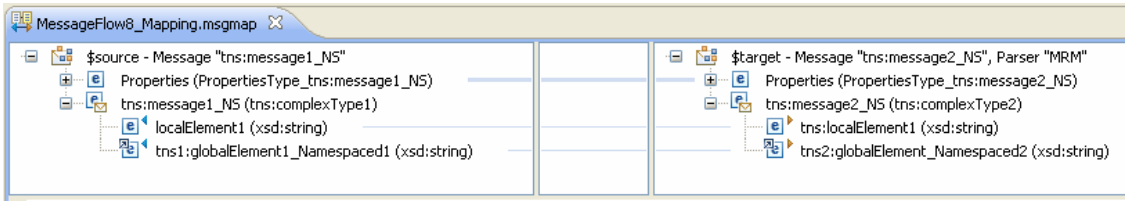


Figure 22 MessageFlow8_Mapping

Map Script	Value
MessageFlow8_Mapping	
Parameters	
$target	
Properties	
MessageSet	"MapToESQL_MessageSet"
MessageType	"{http://www.mrmnames.net}:message2_NS"
MessageFormat	$source/Properties/MessageFormat
Encoding	$source/Properties/Encoding
CodedCharSetId	$source/Properties/CodedCharSetId
Transactional	$source/Properties/Transactional
Persistence	$source/Properties/Persistence
CreationTime	$source/Properties/CreationTime
ExpirationTime	$source/Properties/ExpirationTime
Priority	$source/Properties/Priority
ReplyIdentifier	$source/Properties/ReplyIdentifier
ReplyProtocol	$source/Properties/ReplyProtocol
Topic	$source/Properties/Topic
ContentType	$source/Properties/ContentType
IdentitySourceType	$source/Properties/IdentitySourceType
IdentitySourceToken	$source/Properties/IdentitySourceToken
IdentitySourcePassword	$source/Properties/IdentitySourcePassword
IdentitySourceIssuedBy	$source/Properties/IdentitySourceIssuedBy
IdentityMappedType	$source/Properties/IdentityMappedType
IdentityMappedToken	$source/Properties/IdentityMappedToken
IdentityMappedPassword	$source/Properties/IdentityMappedPassword
IdentityMappedIssuedBy	$source/Properties/IdentityMappedIssuedBy
tns:message2_NS	
tns:localElement1	$source/tns:message1_NS/localElement1
tns2:globalElement_Namespaced2	$source/tns:message1_NS/tns1:globalElement1_Namespaced1

Figure 23 MessageFlow8_Mapping Map Script

The ESQL which is generated includes declarations for the namespace prefixes used in both the input and output messages. The generated ESQL is shown in Figure 24:

Generated_MessageFlow8_Mapping.esql

```
-- This file contains ESQL generated from a msgmap mapping file

CREATE COMPUTE MODULE MessageFlow8_Mapping_Compute

        CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN

            DECLARE xsd NAMESPACE 'http://www.w3.org/2001/XMLSchema';
            DECLARE tns NAMESPACE 'http://www.mrmnames.net';
            DECLARE tns2 NAMESPACE 'http://www.mrmnames2.net';
            DECLARE tns1 NAMESPACE 'http://www.mrmnames1.net';
            SET OutputRoot.Properties.MessageSet = 'L51JS40002001';
            SET OutputRoot.Properties.MessageType = 'message2_NS';
            SET OutputRoot.Properties.MessageFormat = InputRoot.Properties.MessageFormat;
            SET OutputRoot.Properties.Encoding = InputRoot.Properties.Encoding;
            SET OutputRoot.Properties.CodedCharSetId = InputRoot.Properties.CodedCharSetId;
            SET OutputRoot.Properties.Transactional = InputRoot.Properties.Transactional;
            SET OutputRoot.Properties.Persistence = InputRoot.Properties.Persistence;
            SET OutputRoot.Properties.CreationTime = InputRoot.Properties.CreationTime;
            SET OutputRoot.Properties.ExpirationTime = InputRoot.Properties.ExpirationTime;
            SET OutputRoot.Properties.Priority = InputRoot.Properties.Priority;
            SET OutputRoot.Properties.ReplyIdentifier = InputRoot.Properties.ReplyIdentifier;
            SET OutputRoot.Properties.ReplyProtocol = InputRoot.Properties.ReplyProtocol;
            SET OutputRoot.Properties.Topic = InputRoot.Properties.Topic;
            SET OutputRoot.Properties.ContentType = InputRoot.Properties.ContentType;
            SET OutputRoot.MRM.tns:localElement1 = InputRoot.MRM.localElement1;
            SET OutputRoot.MRM.tns2:globalElement_Namespaced2 = InputRoot.MRM.tns1:globalElement1_Namespaced1;

        RETURN TRUE;
        END;

END MODULE;
```

Figure 24 Generated ESQL from MessageFlow8_Mapping

Example 9: Input Body and Output Body REFERENCE, MRM domain

Example 9 demonstrates the mapping of values from an input message parsed in the MRM domain to an output message also in the MRM domain. This time, when invoking the MapToESQL Plugin, check the boxes which indicate you would like to use ESQL REFERENCES for both the Input Body and Output Body. This example demonstrates that the ESQL generated by the Plugin contains REFERENCE declarations which will lead to more performant ESQL code being generated.

Locate the project named `MapToESQL_MessageFlows`, and open the message map named `MessageFlow9_Mapping`. The mapping itself is shown in Figure 25, and the Map Script is shown in Figure 26.

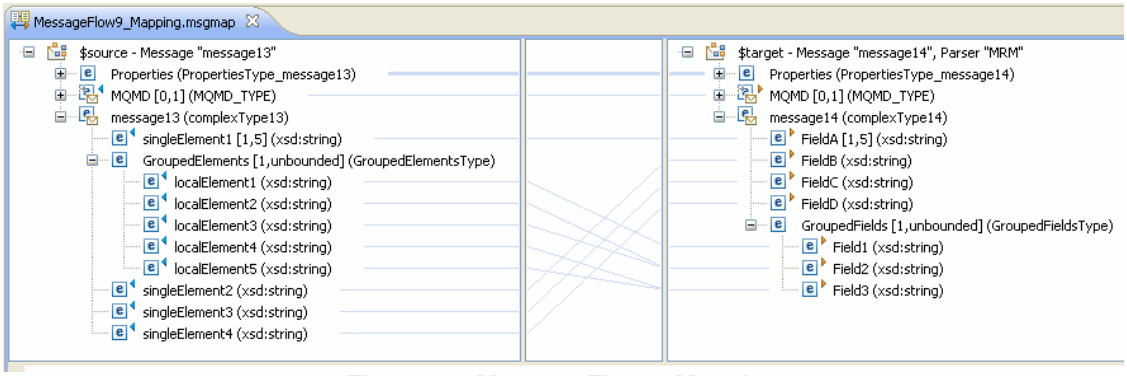
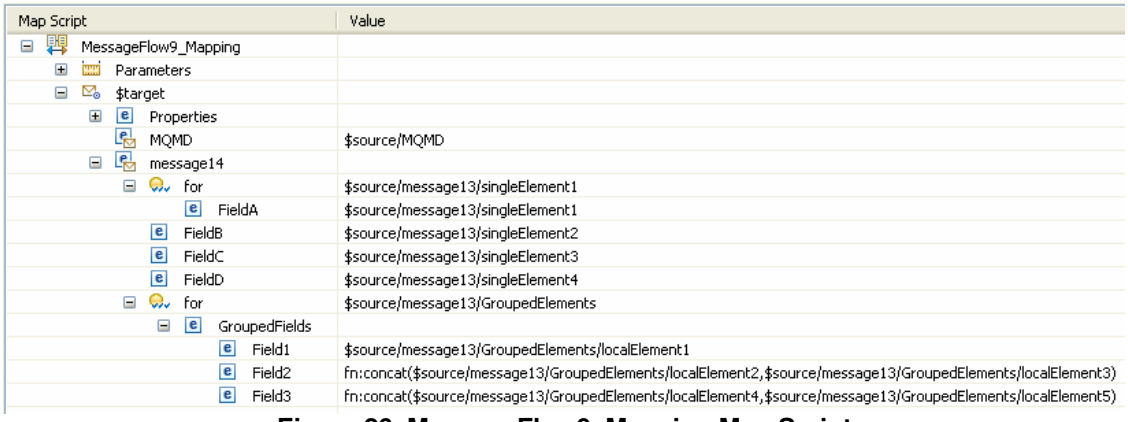


Figure 25 MessageFlow9_Mapping



Map Script	Value
MessageFlow9_Mapping	
Parameters	
$target	
Properties	
MQMD	$source/MQMD
message14	
for	$source/message13/singleElement1
FieldA	$source/message13/singleElement1
FieldB	$source/message13/singleElement2
FieldC	$source/message13/singleElement3
FieldD	$source/message13/singleElement4
for	$source/message13/GroupedElements
GroupedFields	
Field1	$source/message13/GroupedElements/localElement1
Field2	fn:concat($source/message13/GroupedElements/localElement2,$source/message13/GroupedElements/localElement3)
Field3	fn:concat($source/message13/GroupedElements/localElement4,$source/message13/GroupedElements/localElement5)

Figure 26 MessageFlow9_Mapping Map Script

The generated ESQL is shown in Figure 26:

Generated_MessageFlow9_Mapping.esql

```
-- This file contains ESQL generated from a msgmap mapping file

CREATE COMPUTE MODULE MessageFlow9_Mapping_Compute

        CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN

            DECLARE InBodyRef REFERENCE TO InputRoot.MRM;
            DECLARE xsd NAMESPACE 'http://www.w3.org/2001/XMLSchema';
            SET OutputRoot.Properties.MessageSet = 'L51JS40002001';
            SET OutputRoot.Properties.MessageType = 'message14';
            SET OutputRoot.Properties.MessageFormat = InputRoot.Properties.MessageFormat;
            SET OutputRoot.Properties.Encoding = InputRoot.Properties.Encoding;
            SET OutputRoot.Properties.CodedCharSetId = InputRoot.Properties.CodedCharSetId;
            SET OutputRoot.Properties.Transactional = InputRoot.Properties.Transactional;
            SET OutputRoot.Properties.Persistence = InputRoot.Properties.Persistence;
            SET OutputRoot.Properties.CreationTime = InputRoot.Properties.CreationTime;
            SET OutputRoot.Properties.ExpirationTime = InputRoot.Properties.ExpirationTime;
            SET OutputRoot.Properties.Priority = InputRoot.Properties.Priority;
            SET OutputRoot.Properties.ReplyIdentifier = InputRoot.Properties.ReplyIdentifier;
            SET OutputRoot.Properties.ReplyProtocol = InputRoot.Properties.ReplyProtocol;
            SET OutputRoot.Properties.Topic = InputRoot.Properties.Topic;
            SET OutputRoot.Properties.ContentType = InputRoot.Properties.ContentType;
            SET OutputRoot.MQMD = InputRoot.MQMD;
            CREATE LASTCHILD OF OutputRoot DOMAIN('MRM');
            DECLARE OutBodyRef REFERENCE TO OutputRoot.MRM;

            DECLARE A1 INTEGER 1;
            DECLARE A2 INTEGER CARDINALITY(InputRoot.MRM.singleElement1[]);
            DECLARE A1_REF REFERENCE TO InputRoot.MRM.singleElement1[1];
            WHILE A1 <= A2 DO
                MOVE A1_REF TO InputRoot.MRM.singleElement1[A1];
                SET OutBodyRef.FieldA[A1] = A1_REF;
                SET A1 = A1 + 1;
            END WHILE;

            SET OutBodyRef.FieldB = InBodyRef.singleElement2;
            SET OutBodyRef.FieldC = InBodyRef.singleElement3;
            SET OutBodyRef.FieldD = InBodyRef.singleElement4;

            DECLARE A3 INTEGER 1;
            DECLARE A4 INTEGER CARDINALITY(InputRoot.MRM.GroupedElements[]);
            DECLARE A3_REF REFERENCE TO InputRoot.MRM.GroupedElements[1];
            WHILE A3 <= A4 DO
                MOVE A3_REF TO InputRoot.MRM.GroupedElements[A3];
                SET OutBodyRef.GroupedFields[A3].Field1 = A3_REF.localElement1;
                SET OutBodyRef.GroupedFields[A3].Field2 = A3_REF.localElement2 || InBodyRef.GroupedElements[A3].localElement3
                SET OutBodyRef.GroupedFields[A3].Field3 = A3_REF.localElement4 || InBodyRef.GroupedElements[A3].localElement5
                SET A3 = A3 + 1;
            END WHILE;


        RETURN TRUE;
        END;

END MODULE;
```

Figure 27 Generated ESQL from MessageFlow9_Mapping

Example 10: Input and Output support for SOAP domain

Example 10 demonstrates the mapping of values from an input message parsed in the SOAP domain to an output message also in the SOAP domain.

Locate the project named `MapToESQL_MessageFlows`, and open the message map named `MessageFlow10_Mapping`. The mapping itself is shown in Figure 28, and the Map Script is shown in Figure 29.

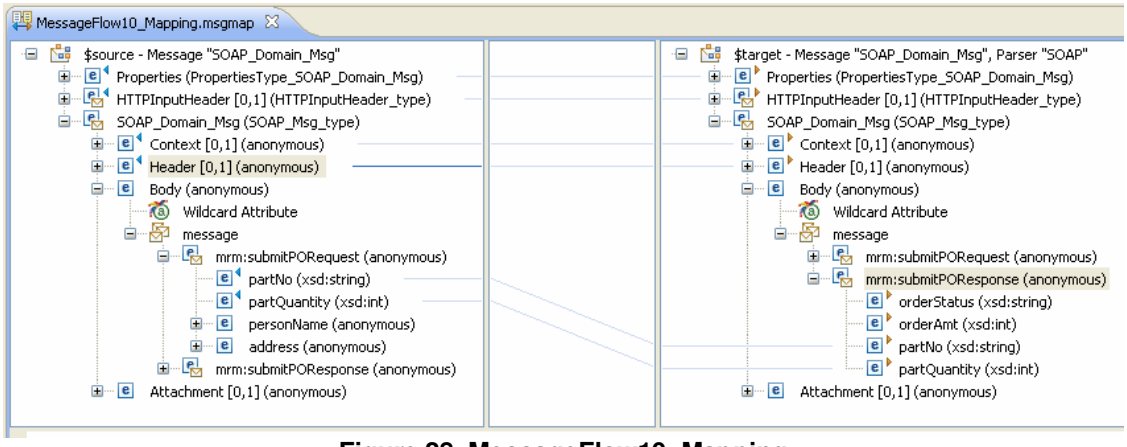
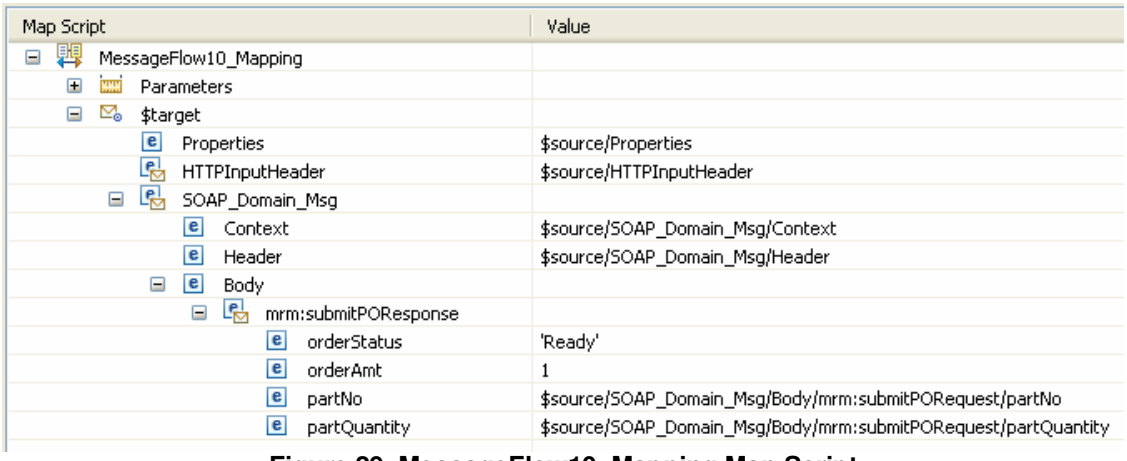


Figure 28 MessageFlow10_Mapping



Map Script	Value
MessageFlow10_Mapping	
Parameters	
$target	
Properties	$source/Properties
HTTPInputHeader	$source/HTTPInputHeader
SOAP_Domain_Msg	
Context	$source/SOAP_Domain_Msg/Context
Header	$source/SOAP_Domain_Msg/Header
Body	
mrm:submitPOResponse	
orderStatus	'Ready'
orderAmt	1
partNo	$source/SOAP_Domain_Msg/Body/mrm:submitPORequest/partNo
partQuantity	$source/SOAP_Domain_Msg/Body/mrm:submitPORequest/partQuantity

Figure 29 MessageFlow10_Mapping Map Script

This time, when invoking the MapToESQL Plugin, select the SOAP domain from both the drop down menus, as shown in Figure 30:

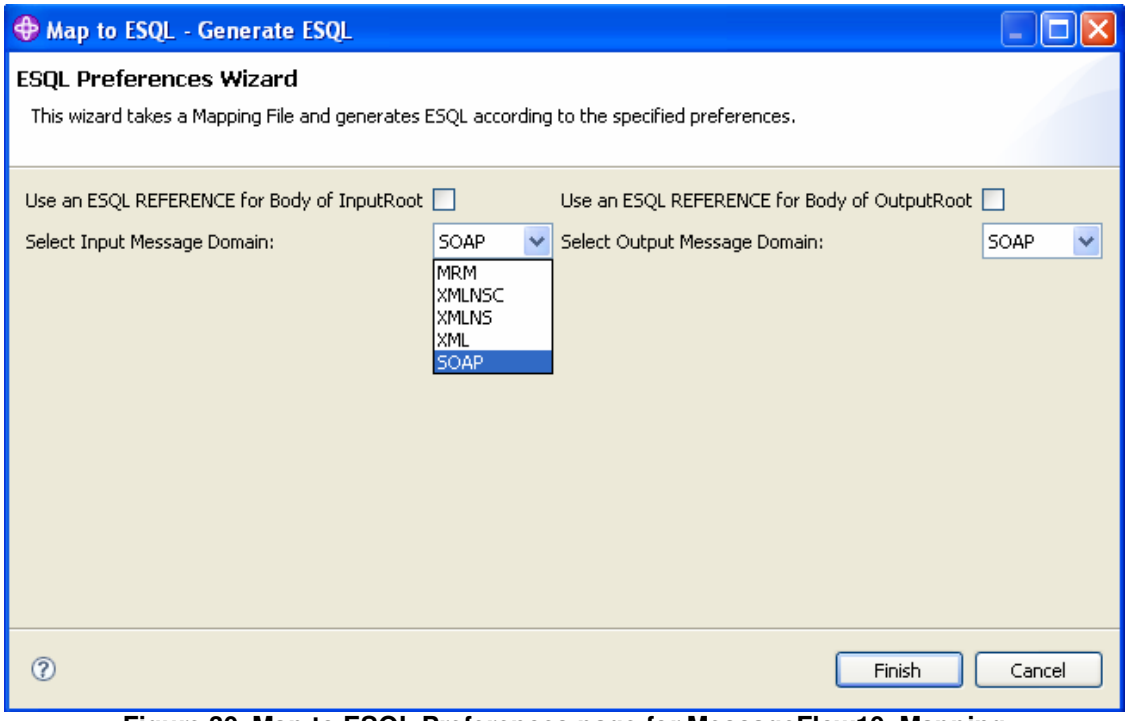


Figure 30 Map to ESQL Preferences page for MessageFlow10_Mapping

The generated ESQL is shown in Figure 31:

Generated_MessageFlow10_Mapping.esql

```
-- This file contains ESQL generated from a msgmap mapping file

CREATE COMPUTE MODULE MessageFlow10_Mapping_Compute

        CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN

            DECLARE xsd NAMESPACE 'http://www.w3.org/2001/XMLSchema';
            DECLARE xsi NAMESPACE 'http://www.w3.org/2001/XMLSchema-instance';
            DECLARE mrm NAMESPACE 'http://www.mrmnames.net/OrderService';
            DECLARE wsdl NAMESPACE 'http://schemas.xmlsoap.org/wsdl/';
            DECLARE soap NAMESPACE 'http://schemas.xmlsoap.org/wsdl/soap/';
            SET OutputRoot.Properties = InputRoot.Properties;
            SET OutputRoot.HTTPInputHeader = InputRoot.HTTPInputHeader;
            SET OutputRoot.SOAP.Body.mrm:submitPOResponse.orderStatus = 'Ready';
            SET OutputRoot.SOAP.Body.mrm:submitPOResponse.orderAmt = '1';
            SET OutputRoot.SOAP.Body.mrm:submitPOResponse.partNo = InputRoot.SOAP.Body.mrm:submitPORequest.partNo;
            SET OutputRoot.SOAP.Body.mrm:submitPOResponse.partQuantity = InputRoot.SOAP.Body.mrm:submitPORequest.partQuantity;

        RETURN TRUE;
        END;

END MODULE;
```

Figure 31 Generated ESQL from MessageFlow10_Mapping

Statement of Support

This second iteration of the Plugin supports the majority of functional requirements of message maps designed using the Broker Toolkit. This section provides a formal statement of support for items which the Plugin should be able to handle. This release has added support for the SOAP domain, and makes the plugin compatible with WMBv7.

Supported Input Message domains: MRM, XMLNSC, XMLNS, XML, SOAP
Supported Output Message domains: MRM, XMLNSC, XMLNS, XML, SOAP

All Folders supported by the Mapping node are supported by the Plugin:

Properties, MQMD, MQCFH, MQCIH, DLH, MQIIH, MQMDE, MQRFH, MQRFH2, MQRMH, MQSAPH, MQWIH, SMQ_BMH, HTTPInputHeader, HTTPReplyHeader, HTTPResponseHeader, HTTPRequestHeader, JMSTransport

For loops are supported.
If constructs are supported.
Conditions are supported.
Else statements are supported.

Calling ESQL functions and procedures from within a map are supported.
Calling XPath String functions (concat, substr) are supported.
Calling XPath Boolean functions (empty, exists, false, not, true) are supported.
Calling XPath Numeric functions (avg, count, max, min, sum) are supported.
Calling XPath Date and Time functions are supported.

Database targets are not supported.

About the Author:

Ben Thompson is a Senior IT Specialist in IBM Software Group EMEA Laboratory Services in Hursley, UK. He has worked with distributed transactional middleware for seven years and has extensive experience designing and implementing solutions using the WebSphere product portfolio with IBM customers worldwide.