MQSeries Integrator V2 – DTD and W3C Schema Import Utilities Version 2.0

10th September 2001

Jim MacNair
IBM
Route 100 MS 1335
Somers, NY
10589

macnair@us.ibm.com

Take Note!

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

Fourth Edition, September 2001

This edition applies to Version 2.0 of *MQSeries Integrator V2 – DTD and W3C Schema import utility* 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 IBM has reviewed each item for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Trademarks and service marks

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

- IBM
- MQSeries
- MQSeries Integrator
- MQSI

The following terms are trademarks or registered trademarks of Microsoft Corporation:

- Windows NT
- Windows 2000
- Microsoft Visual Studio V6
- Microsoft Internet Explorer
- Microsoft

Acknowledgments

The author would like to thank Sean Dunne of the IBM Dublin lab for his assistance in helping the author to understand the workings of the MRM repository database.

Summary of Amendments

Date	Changes
06 June 2001	Initial release
20 June 2001	V1.01 1) Fixed bug where program was not properly detecting duplicate attributes. 2) Added loop detection to avoid stack overflows. 3) Added additional checking for previous errors to suppress further processing.
30 July 2001	V1.02 1) Fixed bug with wrong plugin_key on update of parser type 2) Corrected retrieval of SQLSTATE for non-statement handles.
10 September 2001	V2.0 1) Added W3C schema import utility 2) Upgraded examples to include attributes, additional elements.

Preface

MQSeries Integrator V2 provides a “drag and drop” facility that simplifies the development of message flows In order to use this facility the desired message definitions must be made in the MRM repository. For XML messages, the definition of the message structure is often provided in a DTD or an XML schema. This SupportPac provides an offline utility that will accept a DTD as input and will import the message structure into the MRM repository. A second utility is provided to import a W3C schema into the MRM repository. The imported definition can then be used to build message flows that use the imported information. Although the message sets are stored in the MRM repository, they are not used by the run time broker and should not be assigned to nor deployed to any broker.

The information that is imported into the MRM repository can only be used for the development of message flows. The imported information is not required nor used by the run time broker.

This utility will directly write to the MRM database. Please observe the cautions that are discussed later in this document regarding backup of the MRM database prior to using the import utilities.

The schema import utility supports the W3C standard implementation of XML schemas. No support is provided for other incompatible formats, including the XML Data Reduced (XDR) format. Schemas in the W3C standard format usually have a file extension of “xsd”. Schemas in other, non-standard formats usually have different file extensions.

Bibliography

- *XML Black Book, 2nd Edition,* Natanya Pitts, ISBN 1-57610-783-3.
- *The Complete Reference XML,* Heather Williamson, ISBN 0-07-212734-1.
- *XML Developer's Guide,* Fabio Arciniegas, ISBN 0-07-212648-5.
- *MQSeries Integrator V2 - Using the Control Center*, IBM Corporation. SC33-0826

Chapter 1. Overview

Function 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.

Many transformations involve input and/or output messages in an XML format. In many cases, a definition of the fields in the XML message is available in a Document Type Definition (DTD) file or an W3C XML schema file. MQSI provides a GUI interface that allows compute node transformations to be built using drag and drop operations. However, there is no easy way to import the definitions contained in the DTD or schema file into the metadata repository, so this facility cannot be used. The command line utilities provided with this SupportPac fulfill this need. By allowing the information in the DTD or schema to be imported into the MRM repository database, drag and drop operations can now be used with XML messages.

Warning

For reasons explained later in this document, the DTD and W3C schema import utilities write directly to the MRM database. Because of this, there is the possibility that bugs in the utility can cause corruption of the MRM database. For this reason, it is recommended that the MRM database be backed up before any imports are run.

Chapter 2. Backing up the MRM database

Making a copy of the MRM database

Since the DTD and schema import utilities write directly to the MRM database, the MRM database should be backed up before the utility is used. There are two ways that the utility can be used. The first method is to make a copy of the MRM database under a different name. The import utility than updates the copy and the MQSI MRM import/export utility is used to extract the message from the database copy and then import it into the real MRM database. This technique cannot be use if any of the element or attribute names that are imported contain periods or colons, or if the *InsertAttrs* option is used. The second technique is to update the MRM database directly.

PLEASE MAKE A COPY OF THE MRM DATABASE BEFORE USING THE DTD OR SCHEMA IMPORT UTILITIES. The import utilities write directly to the MRM database. Any bugs in the utilityies could result in the corruption of the MRM database and could leave the configuration manager in an unusable state.

Making a copy of the MRM database.

To make a copy of the MRM database, please use the following steps:

1. Using the MQSI control center, create the desired MRM message sets to contain any imported messages. Stop the control center and the configuration manager.

2. Log on with a user id that has full access to the MRM database.

3. Start a DB2 command window session.

4. Create a directory to contain the backup files. Please use a separate directory for each database that is backed up, and for each backup version.

5. In the resulting DOS command window, change to the desired directory where the MRM database backup files are to be stored.

6. Issue the following command (assuming the MRM database is named MQSIMRDB):

   db2move MQSIMRDB EXPORT

7. If desired, the configuration manager can now be restarted.

8. Start the DB2 Control Center and expand the plus signs until the Databases line is visible.

9. Click on the Databases line with the right mouse button and select Create->Database using Smartguide.

10. Fill in the name of the test MRM database (MRMTEST in this example) in the New Database Name and Database Alias fields. Press the Done button. This operation should take about one minute.

11. Start the DB2 Client Configuration Assistant. Select the new database (MRMTEST) and press the properties button. Click the Register this database for ODBC check box and press the Ok button. Press the close button to exit the Client Configuration Assistant.

12. Return to the DOS command window and issue the following command (assuming the database name of MRMTEST was used): *db2move MRMTEST IMPORT*. This will import the backup copy of the MRM database into the temporary database. The temporary database can now be used to import the DTD and verify that the utility appears to work correctly.

13. Confirm that the messages indicate that the MRM tables are created. Type *exit* to close the DOS command window.

Restoring the MRM database

Before any attempts are made to restore the current MRM database, please make a copy into a new directory. It is also advisable to export the individual message sets and message flows.

1. Stop the control center and configuration manager.

2. Log on with a user id that has full access to the MRM database.

3. Start the DB2 control center and expand the plus signs to make the databases visible.

4. Select the MRM database and click the right mouse button. Select *Drop* from the options and respond *yes* to the confirmation window. The current MRM database is now deleted.

5. Select the Databases tree entry and click with the right mouse button. Select *Create-> Database using SmartGuide*. The database creation smart guide will now be displayed.

6. Fill in the database name and alias name to match the previous MRM database name. Press the *done* button.

7. When the gears stop turning, close the DB2 control center.

8. Start a DB2 command prompt and change to the directory where the database backup was taken.

9. Issue the following command:

   db2move MQSIMRDB IMPORT

   (assuming MQSIMRDB is the name of the MRM database). The database will now be restored.

10. Start the configuration manager and control center.

11. If the database backup was out of date, there may be references in the control center to message sets that no longer exist in the message repository. Please remove these references from the view.

Chapter 3. Installation

Installing the import utilities

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

- id04.pdf Documentation in Adobe PDF format
- license2.txt License file
- mqsiimpdtd.exe DTD import utility
- mqsiimpschema.exe W3C schema import utility
- mqsilistmsgset.exe Utility to list the message sets in the MRM database
- mqsidelmsgset.exe Utility to delete a message set from the MRM database
- dtdparm1.txt Sample parameters file for sample DTD
- impdtd.cmd Command file to import the sample DTD
- mastordr.dtd Sample DTD file
- schparm1.txt Sample parameters file for sample schema
- impschema.cmd Command file to import the sample schema
- mastordr.xsd Sample schema file
- ordrmstr.cpy Sample Cobol copy book
- mastordr.xml Input XML file for sample message flow (XML input)
- data1.txt Input data record for sample message flow (COBOL input)

mqsiimpdtd.exe is the executable program for the DTD import utility, and mqsiimpschema.exe is the executable program for the W3C schema import utility.

mqsilistmsgset.exe and *mqsidelmsgset.exe* are two utilities to display information about the contents of the message repository database and to delete a message set or a possibly incomplete fragment of a message set. The utility programs should be moved to the desired executable directory.

dtdparm1.txt is a sample parameters file for the DTD import utility. It contains the desired options for the utility program, including the name of the MRM database to update, the user id and password to use, the name(s) of the DTD file to import, the name of the Message set to import the DTD into and the name of the root element, among other options.

schparm1.txt is a sample parameters file for the XML schema import utility. It contains the desired options for the utility program, including the name of the MRM database to update, the user id and password to use, the name(s) of the schema file to import, the name of the Message set to import the schema into and the name of the root element, among other options.

The sample parameters file that are provided can be used as templates for the parameters files that must be created to use the utility program.

The remaining files are the sample files that are used in the installation verification example described in detail in Chapter 5. The samples provide a complete working example of the importation and use of a DTD or schema in a message flow.

Chapter 4. Using the DTD importer utility

The MQSI DTD import utility is a single executable program. The utility should be run in a 32-bit Windows environment under either Microsoft Windows NT V4 or Windows 2000. The utility must be run on a system with direct access to the MRM database. This can be the same system that the MRM repository database or a system with a DB2 client that can access the MRM database.

The MQSI configuration manager should be stopped prior to running the DTD import utility. Any message sets to be used by the import utility must be created with the MQSI V2 control center prior to stopping the configuration manager.

Command line parameters

The utility requires a number of parameters to describe the MRM database and the location and name of the DTD file, as well as other required information. Most of these parameters are provided in a parameters file described below. A sample parameters file is included with this SupportPac. The database name and user id can be provided in the parameters file or from the command line. The format of the command is as follows:

```
mqsiimpdtd parameters-file {Userid {Password}} {-v} {-r} {-t} {-d}
```

The first argument is the name of the parameters fileThe user id and password are the user id and password that should be used to access the MRM database. The user id must have update authority against the MRM database. A sample command could be as follows:

```
mqsiimpdtd dtdparms.txt db2admin db2admin -v
```

In addition to the positional parameters described above, the following parameters can be used in any position on the command line:

```
-r or –R    Report only mode – Do not change the database
-v or –V    Verbose – create more detailed messages
-t or –T    Trace – add trace entries to the output messages
-d or –D    Dump dtd info
```

The verbose, trace and dump dtd switches add more detail to the command line output created during the import operation and are generally used for debugging situations.

Format of the parameters file

The parameters file is divided into two sections, namely a header section and a file list section. Comment lines and blank lines are permitted anywhere in the file.

Comment lines must begin with a number or hash sign (“#”), a semi-colon (“;”) or an asterisk (“*”). The rest of the comment line will be ignored.

The header section contains a number of keyword parameters that are required to specify required information such as the name of the MRM repository data base, the name of the schema that the database tables are stored under, the name of the message set that the DTD metadata should be imported into, etc. All parameters in the header section are keyword and value pairs, with an equal sign serving as the delimiter between the keyword and the value. Each parameter must be on a separate line. The header section can be denoted with a [HEADER] delimiter, although this header section identifier is optional. Neither the keywords nor the values may contain embedded blanks.

The file list section is identified by a [FILELIST] section delimiter. This delimiter is required. The file list section contains file names of one or more DTD files. Each file name must appear on a separate line. All the files are concatenated together to produce a single input stream, in the order they are specified in the file list section. All the files are treated as a single DTD.

An XML message with an embedded internal DTD can be used. The <!DOCTYPE specification is not required. However, if a <!DOCTYPE is found in the files, then processing will stop when the matching "]>" delimiter is found. Any entries are the trailing "]>" delimiter will be ignored.

Individual parameters

The following parameters can be used in the header section of the parameters file.

- MessageSet
- MessageName
- RootElement
- MRM_DB
- Schema
- DB_User
- DB_PW
- InsertAttrs
- IgnoreFixAttrs
- IgnoreBadNames
- ChangeBadNames
- ReportOnly
- Verbose
- Trace
- ShowDTDNames

The parameters are not case sensitive and can be specified in all upper case, all lower case or mixed case.

The MessageSet parameter is the name of the MRM message set that will contain the imported information. This message set must be created using the Messages tab in the MQSI Control Center. It must exist before the MQSI DTD import utility can be run.

The MessageName parameter can be used to provide the name that will be assigned to the message that will be created by the import utility in the MRM message repository. If the MessageName parameter is not specified, then the name of the top-level element (RootElement parameter) in the DTD will be used.

The MRM database name (MRM_DB parameter) is the name of the MQSI message repository (MRM) database. The Schema name (Schema parameter) is the name of the schema that the MRM tables were defined under. If the database name and/or the schema name are not known, they can be determined with the DB2 Control Center.

The DB_User and DB_PW parameters can be used to specify the user id and password to be used when accessing the message repository database. The password must match the user id and the user id must have the authority to change the message repository database.

The InsertAttrs parameter controls the generation of identifiers for XML attributes. If this option is selected (set to 'Y'), then the identifier of any attributes in the message set will be prefixed with "(XML.attr)". This prefix will tell the XML parser to render the output as an attribute rather than a child element. This reduces the amount of manual changes that are necessary to the ESQL that is generated. This option also solves the name conflict when an element has the same name as one or more attributes. If this parameter is selected, the message set that the DTD is imported into cannot be exported. The individual elements generated to represent the attributes also cannot be checked out and subsequently checked in. The reason is that the identifiers that are generated for the elements will contain invalid characters. Since the intent of this SupportPac is to provide a build time only facility, this is not believed to represent a problem. If this is in fact a problem, then this option should not be used.

If the IgnoreFixAttrs parameter is set to a 'Y', then any attributes that have #FIXED specified will be ignored. This is useful if these parameters will not be used in message flows.

If the IgnoreBadNames parameter is set to a 'Y', then any attributes and/or elements that have periods or colons in them will be ignored. Any children of any element that is ignored will also be ignored. This parameter is primarily to allow a DTD to be imported into a message set, and the resulting message set can be imported or exported. It is only useful if the elements and/or attributes in question are not being used.

If the ChangeBadNames parameter is set to a 'Y', then any attributes and/or elements that have periods or colons in the name will have the periods and colons changed to underscore characters. This will allow the DTD to be imported and the resultant message set can then be imported or exported. However, if any of the elements or attributes, then the name in the ESQL that is generated must be manually changed to the correct name. If both the ChangeBadNames and the IgnoreBadNames parameters are specified, the ChangeBadNames parameter will have precedence.

The ReportOnly switch will execute the complete utility against the DTD and MRM database. However, it will execute a rollback to undo all changes made to the database. This parameter can be used to test what would happen if a DTD were to be imported into a message set, without actually making any permanent changes to the database. This switch can also be specified on the command line as a "-r" parameter.

The Verbose, trace and showDTDnames parameters will produce additional command line output when the utility is executed. The parameters must begin with either a "Y" for Yes or an "N" for No. The default for each parameter is "N" (No). These parameters are primarily used for debugging situations. These parameters can also be specified on the command line as –v, -d or –t respectively.

DB2 Log file size

The import utility will perform all database operations as a single unit of work. This may require that the size of the DB2 log files be increased, particularly if a large DTD is to be imported.

If the maximum log size is exceeded, error messages should be produced in the application log in the Windows event log. The error messages can be viewed with the Windows event viewer.

To increase the size of the log file, start the DB2 control center and then open the views until the individual databases are visible. Select the MRM database and click on it using the right mouse button. Select the logs tab.

On the Configure Database notebook, on the Logs page, when you click on the Log File Size parameter, the Hint text says the range for NT is [4 - 4095] Pages (one page is 4K bytes). For NT, the range is actually from four to 65535 pages (one page is 4K bytes).

Chapter 5. Using the W3C Schema import utility

The MQSI schema import utility is a single executable program. The utility should be run in a 32-bit Windows environment under either Microsoft Windows NT V4 or Windows 2000. The utility must be run on a system with direct access to the MRM database. This can be the same system that the MRM repository database or a system with a DB2 client that can access the MRM database.

The MQSI configuration manager should be stopped prior to running the import utility. Any message sets to be used by the import utility must be created with the MQSI V2 control center prior to stopping the configuration manager.

Command line parameters

The utility requires a number of parameters to describe the MRM database and the location and name of the schema file(s), as well as other required information. Most of these parameters are provided in a parameters file described below. A sample parameters file is included with this SupportPac. The database name and user id can be provided in the parameters file or from the command line. The format of the command is as follows:

```
mqsiimpschema parameters-file {Userid {Password}} {-v} {-r} {-t} {-d}
```

The first argument is the name of the parameters fileThe user id and password are the user id and password that should be used to access the MRM database. The user id must have update authority against the MRM database. A sample command could be as follows:

```
mqsiimpschema parmsch1.txt db2admin db2admin -v
```

In addition to the positional parameters described above, the following parameters can be used in any position on the command line:

```
-r or –R    Report only mode – Do not change the database
-v or –V    Verbose – create more detailed messages
-t or –T    Trace – add trace entries to the output messages
-d or –D    Dump schema info
```

The verbose, trace and dump schema switches add more detail to the command line output created during the import operation and are generally used for debugging situations. The report only switch runs the entire command and produces all of the resulting command line output and error messages, but does not change the database.

Format of the parameters file

The parameters file is divided into four sections, namely a header section, a file list section, a file location section and a namespace prefix section.. Comment lines and blank lines are permitted anywhere in the file.

Comment lines must begin with a number or hash sign (“#”), a semi-colon (“;”) or an asterisk (“*”). All comment lines are ignored.

The header section contains a number of keyword parameters that specify information such as the name of the MRM repository data base, the name of the schema that the database tables are stored under, the name of the message set that the schema metadata should be imported into, etc. All parameters in the header section are keyword and value pairs, with an equal sign serving as the delimiter between the keyword and the value. Each parameter must be on a separate line. The header section can be denoted with a [HEADER] delimiter, although this header section identifier is optional. Values can be enclosed in a matching pair of single or double quotation marks.

The [FILELIST] section delimiter is required to identify the master schema file. The file list section contains file names of the master schema file.

The [NAMESPACEPREFIX] section is used to provide a mapping between namespaces and the shortened prefix that is used to qualify names. This section is only required if the XML instance documents generated or processed by a message flow will use XML namespaces. If namespaces are to be used in the XML instance documents, the UsePrefixes parameter must be set to "Y" (Yes) in the header section.

When an XML instance document is created, shortened abbreviations for XML namespaces are normally used. XML namespace names should be globally unique and are therefore generally long. A short abbreviation to the full name is usually defined by using the xmlns attribute, usually on the root element. The short abbreviation is then used as a prefix where tag names must be qualified in the XML instance document. For each namespace used in the instance document (generally one per target namespace), a prefix must be defined that will then be included in the names of the appropriate elements and attributes.

Please be aware that the use of XML namespaces is quite complicated, and any further discussion of namespaces is found in the design considerations section of this document. For documentation on XML namespaces (such as there is), please consult various web sites, including the official W3C web site. The author is not aware of a good reference on XML namespaces and would welcome input on this subject.

The [FILELOCATION] section addresses the problem of the vagueness of the schemaLocation attributes on include or import statements. The W3C specification defines the schemaLocation as a "hint" as to the location of a schema. This is very vague and has led to a wide variety of text being used in this part of a schema. The import utility needs to be able to resolve a particular include or import statement to a particular file on the local file system. The import utility will first try to use the contents of the schemaLocation attribute as a file name and will attempt to open that file. If the open is unsuccessful, as in cases where the contents of the schemaLocation is not a local file name, then the import utility will attempt to find an entry in the FILELOCATION section of the parameters file.

Each entry in the FILELOCATION section must be on a separate line. Each line must consist of two strings, separated by one or more blanks. The first string must match the value of the schemaLocation attribute for a particular include or import statement. The second string must be the name of a local file that is the desired schema to be included or imported.

Individual parameters

The following parameters can be used in the header section of the parameters file.

- MessageSet
- MessageName
- RootElement
- MRM_DB
- Schema
- DB_User
- DB_PW
- InsertAttrs
- UsePrefixes
- ReportOnly
- Verbose
- Trace
- ShowSchemaNames

The parameters values and identifiers are not case sensitive and can be specified in all upper case, all lower case or mixed case.

The MessageSet parameter is the name of the MRM message set that will contain the imported information. This message set must be created using the Messages tab in the MQSI Control Center. It must exist before the MQSI schema import utility can be run.

The MessageName parameter can be used to provide the name that will be assigned to the message that will be created by the import utility in the MRM message repository. If the MessageName parameter is not specified, then the name of the top-level element (RootElement parameter) in the DTD will be used.

The MRM database name (MRM_DB parameter) is the name of the MQSI message repository (MRM) database. The database schema name (Schema parameter) is the name that the MRM tables were defined under. It has no relationship to XML schemas. If the database name and/or the database schema name are not known, they can be determined with the DB2 Control Center.

The DB_User and DB_PW parameters can be used to specify the user id and password to be used when accessing the message repository database. The password must match the user id and the user id must have the authority to change the message repository database.

The InsertAttrs parameter controls the generation of identifiers for XML attributes. If this option is selected (set to 'Y'), then the identifier of any attributes in the message set will be prefixed with "(XML.attr)". This prefix will tell the XML parser to render the output as an attribute rather than a child element. This reduces the amount of manual changes that are necessary to the ESQL that is generated. This option also solves the name conflict when an element has the same name as one or more attributes. If this parameter is selected, the message set that the schema file is imported into cannot be exported. The individual elements generated to represent the attributes also cannot be checked out and subsequently checked in. The reason is that the identifiers that are generated for the elements will contain invalid characters. Since the intent of this SupportPac is to provide a build time only facility, this is not believed to represent a problem. If this is in fact a problem, then this option should not be used.

The UsePrefixes parameter will control whether namespace prefixes are inserted in front of XML element and attribute names that will be generated.

The ReportOnly switch will execute the complete utility against the schema and MRM database. However, it will execute a rollback to undo all changes made to the database. This parameter can be used to test what would happen if a schema were to be imported into a message set, without actually making any permanent changes to the database. This switch can also be specified on the command line using the "-r" parameter.

The verbose, trace and showSchemaNames parameters will produce additional command line output when the utility is executed. The parameters must begin with either a "Y" for Yes or an "N" for No. The default for each parameter is "N" (No). These parameters are primarily used for debugging situations. These parameters can also be specified on the command line as –v, -d or –t respectively.

Chapter 6. Creating a message flow

Once the import utilities have been used to create entries in a message set in the message repository, the individual messages can now be used with the drag and drop development capabilities of MQSeries Integrator V2.

It is recommended that the sample provided with this SupportPac be used as a learning experience to understand how to use the imported metadata. Detailed instructions are provided in chapter 9 (Installation verification).

Generating a message transformation using the imported metadata

In order to use the imported metadata, a message flow must be created and one or more nodes must be added to the flow, using the standard MQSI graphical tooling. The nodes must include one or more compute nodes. Select the desired compute node and then press the right mouse button. Select properties from the drop down list box.

Select the Copy message headers only radio button. In order to use the drag and drop capabilities to generate a message transformation, a message set and type must be selected for both the input and output messages. To do this, press the button labeled *Add* near the *Input*s label. Choose the message set and message that correspond to the input message. A tree view of the selected message set is now displayed. Click on any plus signs to expand the tree view and reach lower level elements. In a similar manner, click on the Add button near the *Output*s label and select the message set and message that corresponds to the output message. Again, expand any plus signs to drill down to the lower level elements.

Select the input field name and drag it onto the corresponding output field.

There is no provision for assignment of literals to fields or for more complex operations, such as sub-string or concatenation operations. In this case, one or more arbitrary assignment operations should be done, to generate an assignment of the desired field from some arbitrary field. The generated EQSL can then be modified to reflect the desired operation. This can save considerable time and errors, since the field names will be generated automatically.

Considerations when using the generic XML parser with attributes

The generic XML parser will parse input attributes as children of their respective elements. The resulting parse tree will look very similar for an element that is a child of another element and an attribute of the same element. It is possible to distinguish between an attribute and an element via the use of so-called parser-specific bits. In the case of the generic XML parser, the attribute name can be preceded by the expression XML.attr in parentheses. For example, to refer to attribute C of element b who is a child of element a, the following syntax would be used:

```
Set “OutputRoot”.”XML”.”A”.”B”.(XML.attr)”C” = ‘some data’;
```

In order to distinguish between attributes and elements, the import utilities have the ability to insert the characters (XML.attr) in front of the attribute name. When this attribute is then used to generate a transformation, the ESQL that is generated for the above example is:

```
SET “OutputRoot”.”XML”.”A”.”B”.”(XML.attr)C” = something…
```

Notice that the double quotation in front of the last element name (“C”) is in front of the (XML.attr) qualification rather than after it. This needs to be changed manually (this can be done with a single global change operation).

Chapter 7. Design considerations

There are a number of important considerations when using this utility.

Build time only

The utilities included in this SupportPac are designed for the build time environment only. The message definitions that are produced by the utility are to be used with the XML parser supplied with MQSeries Integrator Version 2. The imported message definitions cannot be used at execution time and should never be assigned to a message broker nor deployed.

Message sets that contain messages imported by the DTD or schema import utilities should never be assigned or deployed to a broker.

Although the MRM run time parser can actually generate XML messages as output, and the MRM repository is being used to create message flows, the MRM run time parser is not used by this SupportPac.

Unique identifiers (names)

The MRM requires that all identifiers within a message set be unique. If more than one element or type were to have the same identifier, problems could arise when the message set is used to build a set of C header structures to be used in a problem. There could be conflicting structures with the same name, and that would result in compilation errors.

Since many different DTDs and schemas could use the same element name for different purposes, each DTD or schema should be imported into a separate message set.

In some cases, a single DTD or schema might contain more than one high level message. In this case, all messages can be imported into a single message set. The import utility will reuse existing elements in the message set as appropriate, providing the definition is the same.

The same attribute name can be used with many different elements. Any such attribute will only be defined once and will be reused for all subsequent elements that have an attribute with the same name.

There is one possible conflict. If an attribute and an element have the same name, a name conflict will arise. Both elements and attributes require an entry in the element name space of the MRM message set, and there is no way to allow both to be defined at the same time in the MRM within a single message set. Therefore, DTDs or schemas with an element and an attribute with the same name are not supported. This problem can be circumvented if the insertAttrs parameter is set to “Y”.

Special characters in element and attribute names

XML permits the use of colons (“:”) and periods (“.”) in element and attribute names. ESQL allows the use of colons in variable names, and periods can be used in variable names as long as the variable name is enclosed in double quotations (e.g. “a.b.c”). However, the MRM does not permit the use of either colons or periods in element identifiers.

If a DTD or schema contains element or attribute names that contain periods or colons, the import utility has the option of importing the elements with the colons and/or periods in the names. The resulting message set can be used for the desired drag and drop operations. It cannot be exported or imported using the MRM import/export utilities, and the individual elements with invalid special characters cannot be checked out or checked in. Fortunately, there is probably no need to modify or export/import the imported message set, so this restriction should not present a problem.

Some special options are provided to offer alternatives for processing DTDs and schemas that have elements or attributes with the special characters in them.

The setting “AllowSpecChars=N” can be specified in the parameters file. If specified, the import utility will check for periods and/or colons in element or attribute names and will then reject the DTD or schema if it finds them. This will alert the user to the fact that element and/or attribute names contain colons and/or periods.

The setting “IGNORE_FIX_ATTRS=Y” will ignore any attributes that have the #FIXED attribute specified. In many cases, the use of colons and periods is limited to attributes with fixed values. By ignoring only these entries in the DTD or schema, the rest of the elements and attributes can be imported without encountering any names with colons or periods. For example, the Acord_IFX.dtd file from the Acord organization (standards organization that sets that develops standards primarily for the insurance industry) uses colons in the names of attributes, but the attribute values are fixed. Please be aware that the release candidate versions of the Acord_IFX.dtd file contain an error where one element is a child of its own children.

Special considerations when importing DTDs

Handling of Attribute lists

Attributes will be treated as children of their respective elements, regardless of whether the element is a compound element with other elements as it’s children or is a simple element.

Public and non-parsed Entities

No support is provided for public entities or for non-parsed entities. No attempt will be made to use the web to search for a public entity. Since non-parsed entities may contain any kind of data, such as images, they might cause confusion if they were read in and inserted into the DTD input.

System entities are supported. When a system entity is found, the file containing the entity will be read and parsed. It will then be available to be referenced by other areas of the DTD.

INCLUDE and IGNORE sections

Support is provided for common uses of INCLUDE and IGNORE clauses. This includes the use of simple parameter substitution of the INCLUDE and IGNORE keywords. The processing of INCLUDE and IGNORE clauses occurs after the processing of ENTITY definitions. If an ENTITY definition is included within an IGNORE section, the entity will still be processed. Other definitions within the IGNORE section will be ignored. For example, the following construct would result in the entity test having the value of “first” rather than “second”.

```
<![IGNORE[
<!ENTITY % test “first” >
]]>
<!ENTITY % test “second” >
```

On the other hand, the following entry should be handled properly:

```
<!ENTITY % selitem “IGNORE”>
<!ENTITY % selitem2 “INCLUDE” >
<![ %selitem; [
<!ELEMENT test EMPTY>
]]>
<![ %selitem2; [
<!ELEMENT test2 EMPTY>
]]>
```

The result would be that the definition of the element test would not be used while the definition of the element test2 would be processed.

XML Namespaces and DTDs

A single message set is a single namespace. There is no support for multiple namespaces within a single MRM message set. To get around this limitation, the DTD import utility will normally include the namespace qualifier as part of the MRM element identifier. This will result in a colon character and in most cases one or more periods becoming part of the element identifier. Colons and periods are not valid characters in an MRM identifier. The DTD import utility gets around this restriction by writing directly to the MRM database. However, the message set cannot be exported or imported, and individual elements with illegal characters in their identifiers cannot be checked in. Fortunately, these restrictions do not affect the drag and drop capabilities of the build-time environment.

Special considerations when importing XML schemas

Overview

The W3C XML schema specification offers a large number of powerful options that allow a user to specify many additional characteristics beyond the capabilities provided by DTDs. However, this additional power comes with a cost in terms of complexity.

Due to the complexity of XML schemas and the inherent limitations in the data modeling capabilities of the MRM repository, a number of XML schema features and functions are either ignored or not supported. Whenever possible, the schema feature is recognized by the import utility but simply ignored. The main features of the XML schema specification are broken into three categories below, namely supported features, features that are ignored and features that are not supported.

Many of the restrictions in the MRM data model in MQSI V2.02 and V2.01 will be lifted in a future release of MQSeries Integrator V2.

The import utility expects well-formed XML schema documents. It is not a validation utility. In most cases, a malformed schema document will be rejected with a proper error message. Depending on the precise problem(s) encountered, more obnoxious behavior may occur (such as traps, loops, etc). If the utility is unable to process a particular schema file, please see if the schema file can be loaded into Microsoft Internet Explorer (V5 or later) or another tool that is intended to validate and debug schema documents.

Types of schemas

There are a number of implementations of XML schemas that were made available prior to the actual release of the official W3C Schema specification, including Document Content Descriptions (DCD) and XML Data Reduced. In particular, XML Data Reduced (XDR) has been commonly used. None of the other schema implementations or proposals are compatible with the W3C specification.

The importer in this SupportPac supports the official W3C schema standard. It does not support other formats. In the case of XDR, a migration program is available from Microsoft that will attempt to convert an XDR schema to a W3C standard schema.

Supported features

The first element found in the schema document must be either XML processing information (XML tag starting with “<?xml”) or the schema element. The schema element must be the high level element for the document. Schema tags (such as element, attribute, group, attributeGroup, etc) may be qualified or unqualified.

Element and attribute tags are supported, including names and references. Elements and attributes can be simple types or complex types with either simple content or complex content. Sequence, choice and all are recognized as groups of elements. Element groups and attribute groups are supported.

Extensions are supported to the extent that they add elements and/or attributes to a base type definition. Restrictions are recognized and processed, but most facets are ignored, due to limitations in the MRM data model.

Annotation, documentation and appinfo tags, as well as XML comments can appear at any point in the schema.

The schema data types of anyType and anySimpleType will be accepted. If child elements or attributes are to be used for an element with a type of anyType, then the ESQL that is generated will have to be modified to include the additional naming levels that are needed.

Features that are ignored

Simple data types are recognized but the data type of all generated metadata entries is set to character (string). The run-time XML parser treats all XML data as character (string) type, so there is no way to specify that particular elements or attributes are of a particular type. In a similar vein, sequence, choice and all groupings of elements are recognized but they are basically treated the same.

Most facets are ignored, including pattern, minimum length (minLength), enumeration, duration, encoding, pattern, period, scale, minimum values, and maximum values (minInclusive, maxInclusive, minExclusive, and maxExclusive). Pattern definitions are ignored. There is no function in the MRM repository in MQSI V2.01 and V2.02 that allows facets to be properly supported.

Keys, keyrefs and unique tags (including field and selector) are ignored.

The block and final attributes are ignored.

Notations are ignored. They are generally used in relation to bitmaps, icons, and other graphic types of data. If a notation is encountered, a warning message is issued and the notation is ignored.

Although many features are ignored, in most cases this does not cause a significant impact to the build time environment.

Features that are not supported

The redefines tag is not supported. If a redefines tag is found in a schema file, it will be ignored and a warning message will be issued. The main reason for this is the complexity of redefines coupled with the seeming low usage that the author has observed.

Abstract entries are not supported. The abstract keyword is ignored.

The list tag (list of values) is not supported.

Unions are not supported.

Substitution groups are ignored, including the attributes associated with substitution groups (block and final).

Duplicate names for elements and attributes

Message sets defined in MQSI V2.02 and V2.01 have a single name space for the element identifiers. An XML schema allows many locally defined elements to use the same name, with different definitions of the contents of the various elements. Since the identifier is the name that is used when a message flow is built, XML schemas that have more than one locally defined element with the same name must have all definitions of the element as simple types with no attributes, or based on a simple type with the same attributes. If they are complex types, they must all use the same complex type definition and no attribute can share the same name. The reason for this is that the multiple element definitions will all share a common element definition in the MRM repository.

Schemas also allow the same element name to appear more than once in the same type definition. For example, a sequence could consist of a, b, a, c. There would be four elements in the type, with a being both the first and the third element. In V2.01 and V2.02, the MRM data model does not allow the same element to appear more than once in a type. If an element or attribute appears in the same complexType definition more than one time, a warning message will be issued and the element will be inserted in the type at the first location where it appears.

Namespace usage in the XML parser in MQSI V2

MQSeries Integrator V2 contains a generic XML parser that will parse any well-formed XML message and will create output messages with well-formed XML tags. This parser uses the domain name of "XML". The generic parser is built using a level of the XML4C parser that does not support XML namespaces directly. It will treat a qualified name as a simple name that happens to have a colon embedded in the name. Therefore, messages with qualified tag names can be parsed or created. There is no checking performed regarding the proper use of XML namespaces within the documents.

Namespaces

Limited support for XML namespaces is provided. This includes the use of namespace prefixes in both the XML schema and the target document and support for include and import statements. However, the author does not want to minimize the complexity of namespaces. The author has seen examples such as the following (taken from the document W3C XML Schema: Dos and DON'Ts), which show some of the many "unintended consequences" and "feechurs" that can occur with the use of XML namespaces within schemas and instance documents.

W3C XML Schema allows you to declare elements inside another element:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://example.com">
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="familyName" type="xs:string" />
        <xs:element name="firstName" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

But generally you should avoid this if possible because the above schema does not match the following instance.

```
<person xmlns="http://example.com">
  <familyName> KAWAGUCHI </familyName>
  <firstName> Kohsuke </firstName>
</person>
```

Instead, you have to write it as:

```
<foo:person xmlns:foo="http://example.com">
  <familyName> KAWAGUCHI </familyName>
  <firstName> Kohsuke </firstName>
</foo:person>
```

Not only does this require more typing, it is also a bad use of XML namespace. To avoid this problem, you should write:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://example.com">
  <xs:element name="person">
```

```
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="familyName" />
        <xs:element ref="firstName" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="familyName" type="xs:string"/>
  <xs:element name="firstName" type="xs:string"/>
</xs:schema>
```

Another way to solve the problem is to blindly add elementFormDefault="qualified" to the schema element.

As the above example demonstrates, namespaces introduce many unintended consequences and much complexity.

Namespaces and namespace prefixes are used in both the schema itself and in an XML instance document that references the schema. This can lead to a lot of confusion when discussing namespaces and qualification of element names. The following sections discuss the use of namespaces and namespace prefixes within the schema document itself and the use of namespace prefixes in the instance documents themselves.

Namespace usage in the XML schema document

Namespace prefixes can be used on the tag names used in the schema document. When tag names are processed for elements within the schema document itself, the qualifier will be ignored. Therefore, xsd:element, xs:element and element are all considered the same.

Namespace qualifiers on name, type and ref attribute values can be used. The prefix name must match a prefix definition in an xmlns attribute in the schema element that the element is defined under. The prefix is used to get the full name of the qualifier. In many cases, the full name of the qualifier looks like a URI (web address). However, it can be any arbitrary string as long as it is unique within the document. Once the full name has been determined, the import utility will search for a matching target namespace on either the main schema or an imported schema. Once this match is found, the desired name is located within the matching schema's global elements, attributes or types. This search will be performed using only the name part of a qualified name. No further checking will be performed.

Namespace prefixes in XML instance documents

There is no definition within an XML schema for the prefix that is to be used in a target XML instance document. The choice of prefix is left to the creator of the instance document. The prefix must be defined in an xmlns attribute in the root element (or on the element itself, or its parent or other ancestors). Since the prefix must be known to the build time, it must be specified in the input parameters file. This prefix will then be used to qualify the item name in any instance documents created or parsed.

If namespace prefixes are to be used, then the usePrefixes parameter must be set to "Y" (Yes) in the parameters file. If the usePrefixes parameter is not set or is set to "N" (No), no namespace prefixes will be generated, regardless of other parameters.

In order to specify a prefix name, an entry must be added to the NAMESPACEPREFIX section of the parameters file. Each entry in this section of the parameters file contains two values separated by white space (blanks or tabs) on a single line. The first parameter is the full name of the namespace as specified in the targetNamespace or xmlns attribute. The second parameter is the prefix to be used when generating XML names for elements or attributes found in this namespace. For example, if the prefix jim is to be used for namespace http://www.ibm.com/example the following entry must be added to the NAMESPACEPREFIX section of the parameters file:

[NAMESPACEPREFIX]
http://www.ibm.com/example jim

Locations of external schema files referenced in a schema document

XML schema files may make reference to other schema files. The include and import elements are used for this purpose. Both of these elements should contain a schemaLocation attribute which should point to the external schema file. Unfortunately, the W3C schema specification is remarkably vague on what a schemaLocation entry should specify, and actually defines it as a “hint” for the schema processor. Since there may be no direct pointer to a particular schema file, a section is provided in the input parameters file to resolve the given “hint” and specify the location of the actual schema file reference.

In order to specify a prefix name, an entry must be added to the FILELOCATION section of the parameters file. Each entry in this section of the parameters file contains two values separated by white space (blanks or tabs) on a single line. If the file name contains any blank characters, the name must be enclosed in single or double quotation marks. The first parameter is the full name of the namespace as specified in the schemaLocation attribute, and the second parameter is the file name. The file name indicates where the file can be found on the local file system. For example, if the external file addresses.xsd is to be used for schemaLocation http://www.ibm.com/example/addresses, the following entry must be added to the FILELOCATION section of the parameters file:

[FILELOCATION]
http://www.ibm.com/example/address addresses.xsd

COBOL and C Language name definitions

There are a number of restrictions on the names of data elements and types. The COBOL language name must be all uppercase, no more than 30 characters long and must not be a COBOL reserved word. The C Language name must not be a C language reserved word. The language names are only used if the message set were to be extracted to produce a COBOL copybook or a C header structure.

Most XML names do not meet the criteria for C and COBOL names. Since there is no intention to use the imported DTD metadata to generate C or COBOL language bindings, unique names will be generated for each element, type and length that will be used for the required COBOL and C language names. These names are not used to build messages.

A simple convention has been adopted to satisfy the requirements of the MRM for valid COBOL and C language names for each element and type. For the COBOL language name, the name will be shortened to a maximum of 24 characters and a unique number of up to six digits will be appended to the end of the name. Any underscores in the name are replaced by dashes and the name is translated to upper case. If the first character is not a letter, it will be changed to the letter “C”.

To generate a unique C language name that does not conflict with any C language reserved words, a unique number of up to six digits is appended to the end of the name.

It bears repeating that these generated names are not used in any way and therefore should be ignored.

Parser name for the Message Set

The run time parser name for the MQSI V2 message set will be changed to XML. This change is necessary to generate the proper high-level data names.

Limitations and large DTDs or schemas

Many of the input parameters have maximum lengths. In all cases, the author believes these limits are beyond limits in the underlying products and should not cause any problems. They are enforced so that the import utility will not exceed its internal memory allocations for certain parameters.

- MRM database name 128
- MRM user id and password 63
- Schema name 63
- Message set name 127
- Root element name 255

The following default limits apply to the number of elements, attributes and entities. Using the corresponding settings in the parameters file, these limits can be increased if necessary.

- Elements 8192
- Attributes 8192
- Entities 2048

The author has tested the utility with DTDs of over 175,000 bytes long containing over 1000 elements and 1800 attributes. There was considerable room left in all internal tables and structures used by the utility. The utility is linked with a 16 MB stack size, which is probably more than necessary since the stack is not heavily used.

OAGIS Business Object Definitions (BODs)

The open applications group has developed definitions of a number of common business objects. They have also defined a number of XML formats for these objects. These definitions are available as DTDs that can be downloaded from the open applications group website (http://www.openapplications.org). There are close to 200 definitions available as of this writing and more are being added.

There is one technique that is used in the BODs that can cause some problems if it is not understood. Some items have different meanings but clearly have a common layout. For example, there are many examples of things like amounts or dates. Both amount billed and unit cost are of type amount, while date ordered and date shipped are of type date and would therefore share a common layout, including sub-fields such as month, day and year. Most XML DTDs will define elements for each unique type of amount or date, giving each element a unique name. In our example above, element names such as AmountBilled, UnitCost, DateShipped and DateOrdered might be defined. In the case of OAG defined entities, all dates would have an element name of DateTime. Each individual DateTime element would then have an attribute that would indicate the meaning of the particular DateTime field.

The use of a common element name can result in a seemingly invalid repetition of a particular element in a group. Within the actual DTD, unique entities are used to identify the particular kind of date, amount, etc that is being referred to. However, once the entity references have been resolved, the result can result in elements like DATETIME appearing many times in a single compound type definition. This is generally regarded as illegal, since a reference to a field name is not sufficient to distinguish which element the reference should apply to.

In the actual DTD, a type might be defined as follows:

<!ELEMENT SHIPSCHDLN ((%DATETIME.CUMULATIVE;)?, (%DATETIME.DELIVACT;)?, (%DATETIME.ENGCHG;)?, (%QUANTITY.ITEM;)?, (%DATETIME.PO;)?,…)

After the entities are resolved, the element (slightly simplified) would look like:

<!ELEMENT SHIPSCHDLN (DATETIME, DATETIME, DATETIME, QUANTITY, DATETIME,…)

To refer to the date of the purchase order, the fourth DATETIME element would have to be referenced. This makes it appear that there are four dates for the purchase order, when in fact the first three DATETIME elements are not dates of purchase orders.

The importer will import the OAGIS DTD definitions, but some care in using the results is recommended. Error messages will be produced indicating that certain elements such as DATETIME elements were found more than once in the same compound type. This message is produced because the MRM does not support the same element appearing more than once in a compound type. Fortunately, for the purposes of using the results for drag and drop operations, this is not a serious limitation. In the example given above, a single DATETIME element will be created and will be inserted at each point in the compound type SHIPSCHDLN.

Alternate design considerations

An alternate design consisting of a DTD import utility that generated a COBOL copybook or C structure and then used the existing COBOL or C language importers was considered. However, these importers reject reserved words in either C or COBOL. Furthermore, the COBOL importer translates all names to uppercase and truncates them to 30 characters. There does not appear to be a simple solution for the reserved word problem. Therefore, the decision was made to directly update the MRM database.

Chapter 8. Installation Verification

Description of the sample DTD or schema environment

A set of sample input data and corresponding DTD and schema definitions have been provided to allow for testing of the import utilities with a sample run-time scenario. The sample environment provides a COBOL copybook and a corresponding data file that can be used to verify the correct operation of the import utilities. A sample DTD and schema file have also been provided. The sample scenario demonstrates a realistic example where an input message in a COBOL copybook format is transformed to an XML message defined using either a DTD or an XML schema file.

The COBOL copybook should be imported into the MRM and used to define the input message in a simple message flow. The DTD or schema is then imported into the MRM and used for the output message. The ESQL statements in the compute node that are necessary for the transformation can be generated using the drag and drop capability of the control center, since both the input and output messages are defined in the MRM.

The sample message flow defines a message that contains a simple order. The order consists of a fixed portion that contains the customer information and order number. The second part of the message consists of one line item entry for each item ordered. The line item entries can occur from 1 to 99 times.

The message flow to be developed will convert the message from a fixed COBOL structure to a corresponding XML message. The XML data format is specified in both a DTD and an XML schema file.

Using the sample environment

The first step is to create a back up of the MRM metadata repository database. This is a precaution in case an error occurs during the import steps and the MRM repository is left in an inconsistent state. A message set should then be defined to contain the results of the DTD import step. The configuration manager should then be stopped and one of the import utilities should be executed. The configuration manager should then be restarted and the COBOL copybook should be imported. The message flow can then be created, and the input and output message definitions should then be added to the compute node. The drag and drop capabilities of the MQSI Control Center can now be used to create the EQSL statements in the compute node that perform the message transformation. The message flow can then be assigned to an execution group and the COBOL copybook message assigned to a broker, and the resulting broker deployed. The sample message data can then be used to invoke the message flow.

Backing up the MRM Repository

The following steps should be followed to insure that a backup copy of the MRM.

1. Stop the Configuration Manager.
2. Start a DB2 Command window.
3. Change to a directory where the backup files from an export of the MRM repository database can be stored. For example, the author created a directory called “\MRM\BACKUP” and used the following command to make that directory the current directory “CD \MRM\BACKUP”.
4. Enter the following command: “db2move MQSIMRDB EXPORT”, where MQSIMRDB is assumed to be the name of the MRM repository database. This command will create a number of files with the entire contents of the MRM database

The MRM database can then be restored by first using the DB2 Control Center to drop and then recreate the MRM database, and then restore the database using the technique above except that the command to restore the database is “db2move MQSIMRDB IMPORT”, where MQSIMRDB is assumed to be the name of the MRM repository database.

Create the Message Set for the XML message definition

With the Configuration Manager running, the Control Center should be started and a message set should be created to contain the results of the DTD or schema import utility. It may also be desirable to create a message set for the results of the COBOL copybook importer. The names of the message sets are not important. The author used the name OrderXML for the XML message definition and OrderMaster for the COBOL message. The COBOL copybook message and the XML message must be in separate message sets, since different run-time parsers are used.

Take note of the message set identifier for the message set that contains the COBOL copybook definition. It will be a strange-looking 13-character field.

Importing the DTD or XML schema file

If the Configuration manager is running, is should be stopped prior to running the XML import step. The Configuration Manager caches certain information in memory and therefore does not notice when a DTD or schema file is imported and the repository database is directly updated.

Execute the DTD or schema command line importer utility, using the impdtd.cmd or impsch.cmd file that is provided with this SupportPac. If necessary, the impdtd.cmd or impsch.cmd file should first be edited to match the local environment. Similarly, the appropriate parameters file (parmdtd1.txt or parmordr.txt) should be edited to match the local environment. In particular, the names of the database, user id, password, database schema name and message set name should be checked and corrected if necessary. A report file called output.txt will be produced, documenting the results of the import step. Check the output file (notepad should work) and ensure that the import step was successful.

Common errors in this step include incorrect database, database schema names, user ids and passwords.

Restart the configuration manager and then start the MQSI Control Center.

Importing the COBOL copybook and creating an input message

Using the MQSI Control Center, import the COBOL copybook into the MRM repository and create a message definition, using the following steps.

1. Select the Message Sets tab and then the desired message set that is to contain the COBOL message.
2. With the mouse pointer on the message set, click the right mouse button and select check out. Again click the right mouse button and select import to Message Set->COBOL.
3. Press the Browse button and navigate to the COBOL copybook file (ordrmstr.cpy). Select this file and then press the Finish button. The COBOL copybook should now be imported into the selected message set. Click on the types button with the right mouse button and select Add to Workspace->Compound Type. Select the ORDER_MASTER_TYPE entry and press finish.
4. Select the Messages item in the message set and press the right mouse button. Select Create with SmartGuide->Message. Type the name of the new message (OrderMaster) and then click on the Type list box. Select the ORDER_MASTER_TYPE and then press Finish. The message should now be created, using the ORDER_MASTER_TYPE for its layout. Check in the new message.

Creating a Message Flow

1. Select the Message Flows tab. Click on the Message Flows entry with the right mouse button and then select Create->Message Flow. Give the message flow a name (OrderMasterDemo) and then click the Finish button. The new message flow will now be created.

2. Expand the IBM Primitives selections and drag an MQInput node, two MQOutput nodes and a compute node to the right hand panel.
3. Rename the MQOutput Nodes to MASTER.OUT and FAILURE respectively. Open the properties and change the queue names to MASTER.OUT and FAILURE respectively.
4. Create a connection from the failure terminal of the input node to the input node of the FAILURE MQOutput node. Connect the out terminal of the MQInput node to the input terminal of the compute node and the out terminal of the compute node to the input terminal of the MASTER.OUT node.
5. Rename the MQInput node to MASTER.IN. Change the properties of the MQInput node so that the input Queue name is MASTER.IN. Select the defaults tab and change the domain to MRM. Select the name that corresponds to the identifier of the COBOL copybook message set (the weird 13 character field). Type the name of the message in the message type field (OrderMaster) and change the Message Format default to CWF (Custom Wire Format).
6. Rename the compute node to CobolToXML.

Using the Drag and Drop capability of the MRM to create the ESQL statements

The drag and drop capabilities of the MRM will now be used to create the ESQL statements that will map the input COBOL copybook fields to the output XML tag values.

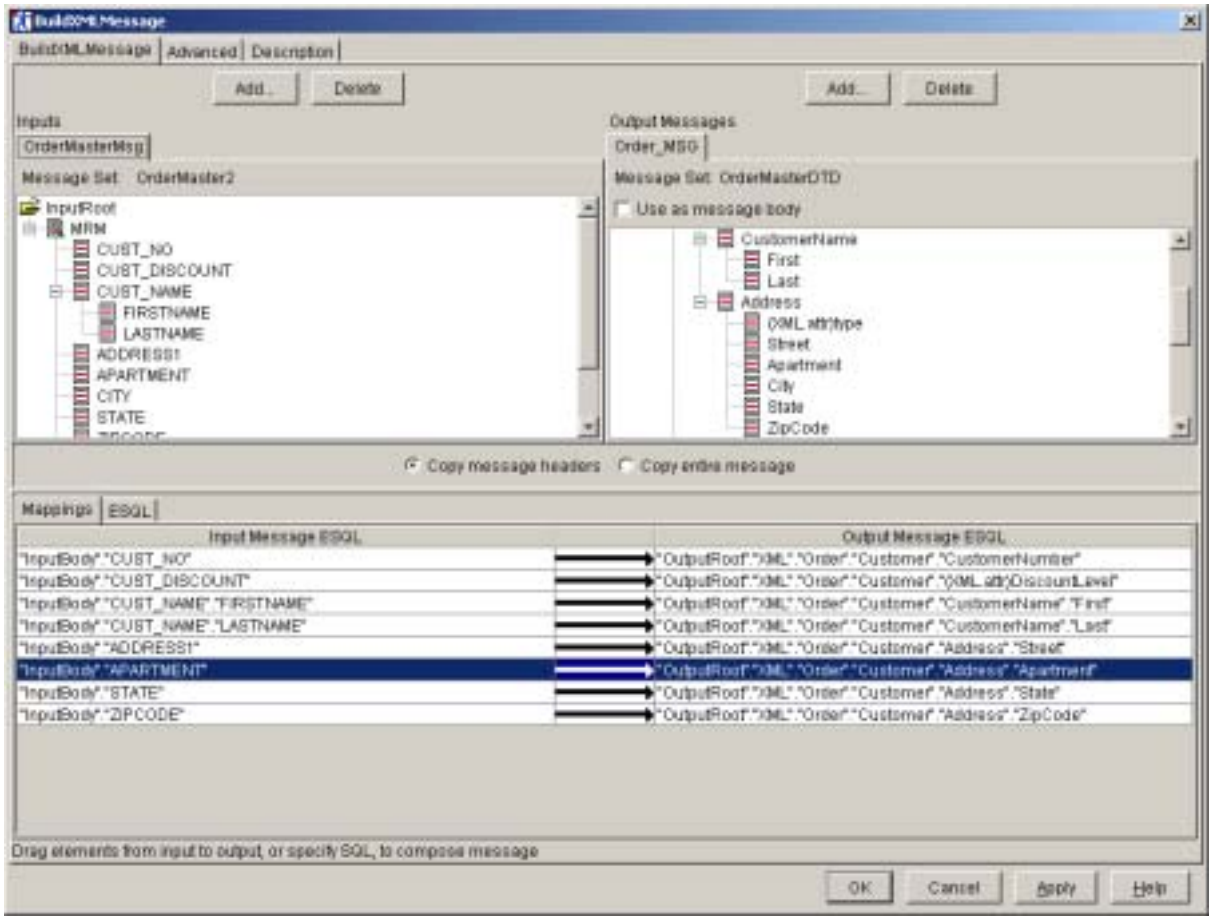
1. Open the properties window of the compute node. Check the Copy message headers radio button. Select the Mappings tab (if necessary). Press the add button near the Inputs label. Select the message set and message that correspond to the input COBOL copybook message (OrderMaster message in the OrderMaster message set, if the suggested names were used). Click on the add button near the Output Messages label and select the message set and message that correspond to the imported DTD message (OrderXML and Order_MSG respectively if the suggested names were used).
2. Click on the plus signs to open up the individual field names. Select the input field name and drag it by using the left mouse button onto the corresponding output field name. A corresponding ESQL SET statement should now be generated. The suggested mappings are given below.

Field mappings for demo programs

COBOL copybook name	XML name
CUST_NO	CustomerNumber
CUST-DISCOUNT	(XML.attr)DiscountLevel
FIRSTNAME	First
LASTNAME	Last
‘US’	(XML.attr)type
ADDRESS1	Street
STATE	State
CITY	City
ZIPCODE	ZipCode
ORDER_NO	OrderNumber
ITEM_NO	Item
QTY	Quantity
PRICE	Price
COST	Cost
WEIGHT	(XML.attr)weightLBs
COLOR	(XML.attr)color
SIZE	(XML.attr)size
SHIP-CODE	(XML.attr)shipby
CARRIER-CODE	(XML.attr)carrier
DESCRIPTION	Description

Since the address type is a literal of 'US', this statement cannot be generated directly using the drag and drop capabilities. It is usually easier to generate a mapping to some arbitrary field, such as the LINE-COUNT fields, and then modify this mapping to replace the LINE-COUNT reference with a literal of 'US'.

When the drag and drop process is partially completed, the screen should look something like the following:



Final tailoring of the generated ESQL

The drag and drop creates most of the ESQL that is necessary to perform the desired transformation. However, there is still some "fix-up" that is necessary, to handle things like the loops that are necessary for items that occur more than once. To perform this tailoring, select the ESQL tab. Select the last four generated statements and cut them out and paste them below the warning line. Change them so that they appear as follows:

```
-- Enter SQL below this line.  SQL above this line might be regenerated, causing any modifications to be lost.
SET "OutputRoot"."XML"."Order"."Customer"."Address"."type" = 'US';
SET I = 1;
WHILE I <= CAST("InputBody"."LINE_COUNT" AS INTEGER) DO
        SET "OutputRoot"."XML"."Order"."Item"[I][I]."ItemNumber" = "InputBody"."ORDER_ITEMS"[I][I]."ITEM_NO";
        SET "OutputRoot"."XML"."Order"."Item"[I]."Quantity" = "InputBody"."ORDER_ITEMS"[I]."QTY";
        SET "OutputRoot"."XML"."Order"."Item"[I]."Price" = "InputBody"."ORDER_ITEMS"[I]."PRICE";
        SET "OutputRoot"."XML"."Order"."Item"[I].(XML.attr)"color" = "InputBody"."ORDER_ITEMS"[I]."COLOR_CODE";
        SET "OutputRoot"."XML"."Order"."Item"[I].(XML.attr)"size" = "InputBody"."ORDER_ITEMS"[I]."SIZE_CODE";
        SET "OutputRoot"."XML"."Order"."Item"[I]."Cost" = "InputBody"."ORDER_ITEMS"[I]."ITEM_COST";
        SET "OutputRoot"."XML"."Order"."Item"[I].(XML.attr)"shipBy" = "InputBody"."ORDER_ITEMS"[I]."SHIP_CODE";
        SET "OutputRoot"."XML"."Order"."Item"[I].(XML.attr)"carrier" = "InputBody"."ORDER_ITEMS"[I]."CARRIER_CODE";
        SET "OutputRoot"."XML"."Order"."Item"[I]."Description" = "InputBody"."ORDER_ITEMS"[I]."DESCRIPTION";
        SET I=I+1;
END WHILE;
```

The author finds the easiest way to make perform the final tailoring is to use a blank notepad session. Start a blank notepad session (type notepad at a DOS prompt and press enter). Select the entire block of EQSL by selecting all of the text in the ESQL window by holding down the ctrl key and pressing the “a” (select all) and then the “c” (copy to clipboard) keys. Switch the focus to the notepad session and then hold down the ctrl key and press the “v” key (paste from clipboard) to move the entire block of ESQL statements to the notepad session. Maximize the notepad session.

Insert a statement to set the address type to a constant of “US”. Perform a global change to move the double quotations from the front of the (XML.attr) prefixes to the end (on Windows 2000, hold down the ctrl key and press “h”). Insert the looping control statements as shown above and then insert the subscript identifiers on the item elements. Again, this can all be done with a single global change (of “Item” to “Item”[I] and "ORDER_ITEMS" to "ORDER_ITEMS"[I]).

Finally, select the entire block of ESQL in the notepad session (on Windows 2000 hold down the ctrl key and press the “a” key, or on Windows NT use the menu item to select all) and copy it to the clipboard (hold down the ctrl key and press the “c” key). Switch the focus to the control center, make sure the entire block of ESQL is selected and paste the modified EQSL in to replace the original ESQL (hold down the ctrl key and press the “v” key).

The key changes made above are to insert the looping statements (SET statements for the loop variable “I” and a WHILE statement to control the looping), and the insertion of index field entries for each variable that can occur a variable number of times.

The statement that was generated to set the address type attribute must be changed to set the attribute to the literal ‘US’.

Finally, all attribute references must be changed to move the double quotation from the beginning of the attribute prefix to the end of the attribute prefix. One way to do this is to start an empty notepad session. Select the entire ESQL block by holding down the ctrl key and pressing “a”. This should mark all the EQSL in the compute node. Copy the ESQL to the clipboard by holding down the ctrl key and pressing “c”. Switch the focus to the empty notepad session and paste the ESQL statements into the notepad by holding down the ctrl key and pressing “v”. Perform a global change in the notepad session replacing “(XML.attr) with (XML.attr)”. This will move the double quotations to the proper location. Select the entire block of ESQL, copy it to the clipboard by holding down the ctrl key and pressing “c”. Finally, switch the focus to the control center, select the entire block of ESQL (ctrl-A) and then paste the modified ESQL into the properties window.

Execution of the Message Flow

The message flow can now be assigned to an execution group and deployed using the following steps:

1. Check in the Message flow.
2. Select the Assignments tab and check out the desired message broker. Select the message set (OrderMaster) for the COBOL copybook and drag it to the desired message broker. Select the new message flow and assign it to the desired execution group. Check in the message broker. Please do not assign the message sets used for the XML DTD or schema imports.
3. Use the MQSeries explorer (or runmqsc) to create the necessary local queues (MASTER.IN , MASTER.OUT and FAILURE).
4. Start the desired broker if necessary. Select the broker and click the right mouse button. Select Deploy->Delta Assignments Configuration. Press OK to acknowledge the message and monitor the log panel to check for the completion of the deployment operation. If necessary, correct any errors and rerun the deploy operation.
5. Using a utility such as RFHUTIL or MQSIPUT, read in the data1.txt sample message data and write it in a message to the MASTER.IN queue. Verify that the target message has been generated and is on the MASTER.OUT queue (again, use the RFHUTIL utility to read the XML message from the output queue and display it in an XML format).

Chapter 9. Problem Determination

Reporting problems

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

- Description of the problem
- General description of the environment
- Copy of the DTD(s) or schema files and the utility parameters file that lead to the problem – please include all files that are needed to process the DTD or schema, including any references to other files, such as system entities in DTDs or files referenced in include or import statements in XML schemas.
- If the problem involves database issues, an exported copy of the repository database. To create this, create a temporary directory and issue the db2move command using the database name and the word EXPORT as the two parameters. Please zip the files that are created in the directory into a single file and include that with the problem report.

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

Unable to connect to MRM database

MRM connection problems are most often caused by incorrect values in the parameters files. Check the database name, database schema name, userid and password parameters. Use the mqsilistmsgset utility (part of this SupportPac) and see if it can connect to the database. Make sure that an ODBC connection has been defined (use the DB2 client configuration assistant).

Could not locate message set xxxxxx

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

Stack Overflow

Stack overflows should not occur. The utility does not make heavy use of the stack and the size of the stack has been increased from the normal default.

"Loop detection" was added in version 1.01. The stack size was increased and a maximum depth check was added. If the program finds itself more than 50 levels down in a DTD, it will terminate and display a list of the element names that have resulted in reaching the maximum allowed depth. The element names should be checked for duplicate names. This would indicate a loop in the input DTD. The loop should be fixed. Should any user have a DTD that has 50 or more legitimate levels, please contact the author.

Problems with the repository database

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

If XML elements or attributes are imported, and the names contain periods or colons, or if the InsertAttrs option is selected in the parameters file, any attempts to export the message set or to

check out and check in an element with such a name will fail. This is normal behavior and does not indicate a problem with the database structure.

MRM repository background

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

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

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

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

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

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

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

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

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

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

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

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

XML messages can only contain string data. As a result, all simple elements inserted into the repository database by the import utility will be of type string. Other element types can be encountered if the elements are imported by the C or Cobol importer or defined within the Control Center GUI.

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

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

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

Diagnostic utilities

Two command line utilities for the MRM are provided with this SupportPac. The first will display a list of the messages and message sets defined in the MRM repository. The second will delete a message set.

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

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

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

```
mqsilistmsgset MQSIMRDB db2admin db2admin DB2ADMIN
```

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

```
Mqsidelmsgset TestMsgSet MQSIMRDB db2admin db2admin DB2ADMIN
```

Chapter 10. Schema and DTD overview

A comprehensive overview of XML schemas and DTDs is beyond the scope of this document. A brief overview will be provided for those that are new to DTDs and/or XML schemas.

DTDs are covered in most books that cover XML, including the books referred to in the bibliography. XML schemas are fairly new. The last two books cover XML schemas and can serve as a good introduction. There is also a tutorial available at the W3C web site.

XML offers a simple method of providing self-defining data, modeled after the widespread use of HTML in the web. XML documents merely had to follow a few simple rules to be considered "well-formed". To be considered "well-formed", an XML document must follow the basic rules for XML tags, with matching pairs of begin and end tags, proper nesting, and all tag names and data must be character data using the approved character encoding. All tags must begin with a less than sign and end with a greater than sign. Attributes must be separated by white space and consist of name and value pairs, with the value enclosed in matched single or double quotations.

Any tags with any names and any data in any hierarchy could be considered a well-formed XML document, whether it was what was intended or not. In many cases, there is a requirement to be able to define what types of data should be allowed in an XML document, especially what tag names should be allowed within any particular part of an XML instance document. The definition of what the data should look like is called metadata, while any actual example of the data defined by the metadata is called an instance document.

There have been two major efforts to define structures that allow the metadata for an XML document to be specified. The first effort was included in the XML 1.0 specification and is known as the Document Type Definition (DTD). The DTD specification allows elements and attributes to be defined, as well as the document hierarchy. DTDs have been well accepted and widely used.

XML DTDs have been perceived as having a number of shortcomings. As a result, a major effort was undertaken to produce a standard method to define metadata for XML documents. The result was the W3C standard for XML schemas. This standard is quite new, being released in 2001. The schema standard is not based on the previous DTD specification.

XML schemas themselves are well-formed XML documents. In addition to providing more capability to describe the hierarchy of a target XML document, XML schemas include a lot of support to define and restrict the types of data that can be contained in XML elements and attributes. XML schemas also include support for XML namespaces. The schema standard offers considerably more capability but as a result is considerably more complex.

The next section will now show a simple DTD and explain some of the key characteristics, followed by a similar section covering XML schemas.

DTD basics

A DTD contains a set of specialized entries to define the elements, attributes and hierarchy of an XML instance document. The three most commonly used entries include elements, attribute lists and entity definitions. There are a number of less commonly used entries that are supported by the DTD importer but which will not be covered here.

Each element that may be used in an instance document must be defined with a element entry. The element entry gives the name of the element and the type of data that is allowed. The data can consist of character data only, child elements only or a combination of character data and child elements (usually referred to as mixed content). The child elements can be in a sequence or can require a choice, and can be specified as being optional or mandatory, and occurring a maximum of one time or any number of times. In the following samples, the first element is a simple element with character data only. The second element consists of three child elements in a particular order. The third element can contain character data as well as three child elements. The fourth element can

contain any number of 3 child elements in any order. The fifth example defines an element that has an optional child of Comment, followed by a child named AccountNo, one or more children named Street and child elements named City, State and Zipcode.

```
<!ELEMENT test1 (#PCDATA)>
<!ELEMENT test2 (city, state, zipcode)>
<!ELEMENT test3 (#PCDATA|city|state|zipcode)+>
<!ELEMENT test4 (Comment|itemNumber|Cescription)+>
<!ELEMENT optional (Comment?, AccountNo, Street+, City, State, Zipcode)>
```

Attributes can be associated with an element by using the ATTLIST entry. The ATTLIST entry starts with the name of the element that the attributes apply to. This is followed by one or more attribute definitions. The attribute definitions consist of the attribute name, the attribute type and attribute characteristics.

For example, the following example associates an attribute named type with an element named date, and three attributes named partno, shipto and shipby with an element named item. The partno attribute is of type ID, which means it must contain a value that is different from all other ID attributes defined in a particular instance document.

```
<!ELEMENT date (#PCDATA)>
<!ATTLIST date type CDATA #REQUIRED>
<!ELEMENT item (itemNo, Quantity, Price, Comment?)>
<!ATTLIST item partno ID #REQUIRED shipto CDATA #IMPLIED shipby CDATA #REQUIRED>
```

Entities consist of replacement text that is referred to with a name enclosed between a percent sign and a semi-colon. Entities can be defined locally or can be brought in from another file, either locally on the system or remotely over a network (known as SYSTEM and PUBLIC entities respectively). Entity values can contain other entities.

The following example shows first the definition of two entities and then examples of their use.

```
<!ENTITY % elem1 '<!ELEMENT City (#PCDATA)>
<!ENTITY % attr1 'type CDATA #REQUIRED'>
%elem1;
<!ATTLIST City %attr1;>
```

Schema basics

An excellent primer on XML schemas is available at the W3C web site. The author recommends this primer for anyone who wants to learn what XML schemas are and how to use them. The following few paragraphs will discuss a few of the most common aspects of XML schemas.

XML schemas are themselves well-formed XML documents. All the information is contained in individual attributes and in the hierarchy of the schema document itself. Schemas use XML elements to define elements, attributes and types that are used in the target instance documents.

An XML schema must have a top-level element of type Schema. The schema element itself has attributes related to the rest of the schema, primarily namespace declarations. Individual elements, types and attributes can be declared globally (items whose parent is the schema element) or locally. Items that are declared globally must have a unique name, and can be referenced by other declarations within the schema. Items that are declared locally need not have unique names but cannot be referenced elsewhere within the schema document.

Elements that appear in the target instance documents must be declared with an element tag. Two examples of element declarations are given below, one for a global element named ZipCode that is referenced by a local element declared as a child within the Address element. The ZipCode element must be a positive integer with a value greater than zero and less than or equal to 99999. The Street element must occur from one to three times, and the apartment element is optional.

```
<xsd:schema xmlns:xsd= "http://www.w3.org/2001/XMLSchema">
<xsd:element name=”ZipCode”>
 <xsd:simpleType>
  <xsd:restriction base=”xsd:positiveInteger”>
   <xsd:minExclusive value="00000"/>
   <xsd:maxInclusive value="99999"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name=”Address”>
 <xsd:complexType>
  <xsd:element name=”Street” minOccurs=”1” maxOccurs=”3” type=”xsd:string”/>
  <xsd:element name=”Apartment” minOccurs=”0” type=”xsd:string”/>
  <xsd:element ref=”City” type=”xsd:string”/>
  <xsd:element name=”State” type=”xsd:string”/>
  <xsd:element ref=”ZipCode”/>
  <xsd:attribute name=”type” type=”xsd:string”/>
 </xsd:complexType>
</xsd:element>
</xsd:schema>
```

The Address element includes an attribute called type, and child elements called Street, Apartment, City, State and ZipCode. ZipCode and Address are global elements. The XML schema defined above would match an instance document like the following:

```
<Address>
  <Street>1000 Main St</Street>
  <Street>Box 50</Street>
  <City>Anytown</City>
  <State>NY</State>
  <ZipCode>11111</ZipCode>
</Address>
```

Chapter 11. Appendix A. Sample Parameters Configuration Files

DTD import utility sample

A sample parameter file is displayed below. Lines beginning with a '#', '*' or ';' are comments.

```
#
# Parameters file for MQSI V2 DTD import utility
#
#
# Name of the message set to import the message into
#
MessageSet=NewOrderSet2
#
# Name of the MRM database to import into
#
MRM_DB=MRMTEST
#
# Name of the DB2 schema
#
Schema=DB2ADMIN
#
# MRM Database User id - can be overridden on the command line
#
DB_User=db2admin
#
# MRM Database Password - can be overridden on the command line
#
DB_PW=db2admin
#
# Name of the top level element in the MRM message
#
RootElement=Order
#
# Name of the message to be created in the message set
#
MessageName=Order
#
# Whether to include (XML.attr) as part of the element identifier for attributes
#
InsertAttrs=N
#
# Whether or not to ignore attributes with the #FIXED property
#
IgnoreFixAttrs=Y
#
# Debugging options
#
Verbose=N
ShowDTDnames=N
Trace=N
#
# Report only – do not update the MRM database – display messages only
#
ReportOnly=N
#
# List of DTD files appears after the [FILELIST] section header
# Each file name must be qualified if necessary and on a separate line
#
[FILELIST]
mastordr.dtd
```

W3C schema import utility sample

A sample parameter file is displayed below. Lines beginning with a '#', '*' or ';' are comments.

```
#
# Parameters file for MQSI V2 Schema import utility
#
#
# Name of the message set to import the message into
#
MessageSet=OrderMasterSchema
#
# Name of the MRM database to import into
#
MRM_DB=MRM
#
# Name of the DB2 schema
#
Schema=DB2ADMIN
#
# MRM Database User id - can be overridden on the command line
#
DB_User=db2admin
#
# MRM Database Password - can be overridden on the command line
#
DB_PW=db2admin
#
# Name of the top level element in the XML message
#
RootElement=Order
#
# Name of the message to be created in the message set
#
MessageName=Order
#
# Whether to include (XML.attr) as part of the element identifier for attributes (Y or N)
#
InsertAttrs=Y
#
# Whether to insert an xmlns attribute on the root element of the message (Y or N)
#
InsertXmlns=Y
#
# use namespace prefixes on output names (Y or N)
#
UsePrefixes=Y
#
# Debugging options (Y or N)
#
Verbose=N
Trace=N
#
# List of schema files appears after the [FILELIST] section header
# Each file name must be qualified if necessary and on a separate line
#
[FILELIST]
mastordr.xsd

[NAMESPACEPREFIX]
```

```
#
# prefix to use for a given namespace
#
#http://www.example.com/test  exmp

[FILELOCATION]
#
# location of file for a given namespace
#
#http://www.example.com/test testfile.xsd
```

End of Document