MQSeries Integrator
Message tree parser
Version 1.3

23rd April 2001

Neil Kolban
MQSeries Technical Support
IBM Advanced Technical Support
Dallas, TX
USA

kolban@us.ibm.com

Take Note!

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

Second Edition, April 2001

This edition applies to Version 1.3 of *MQSeries Integrator – Message tree parser* and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Notices

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

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

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

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

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

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

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

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

Trademarks and service marks

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

- IBM
- MQSeries
- MQSeries Integrator
- MQSI

The following terms are trademarks of other companies:

- NEON New Era of Networks, Inc.

Summary of Amendments

Date	Changes
December 2001	Initial release
April 2001	Bug fix for character string fields greater than 255 characters

Preface

The IBM MQSeries Integrator V2 (MQSI V2) product provides a wealth of capabilities for formatting and reformatting data. The initial parsing and ultimate generation of data received or put to an MQSeries queue is performed by a parser. IBM supplies a suite of pre-built parsers to accommodate many current data forms. This SupportPac provides an additional parser to complement those supplied with the product.

Chapter 1. Introduction

When data is supplied to MQSI V2 for processing, that data is supplied in the form of an MQSeries message on an MQSeries queue. The developer of an MQSI V2 message flow will have included an MQInput node and associated that node with a queue. When a message arrives on the queue, the MQInput node is activated and retrieves the message. Based on either the implicit content of the message or attributes of the MQInput node, a parser is selected to decompose the message structure into its constituent elements.

As the message passes through MQSI V2, it may eventually reach an MQOutput node. At this stage, the message is placed upon an MQSeries queue. To perform this task, the individual elements of the message are recombined into a binary or wire format. The parser again performs this.

The parsers supplied with MQSI V2 include:

- XML – Extensible Markup Language
- MRM – Fixed length data structures
- NEON – Invocation of NEON input/output formats
- BLOB – Unparsed data

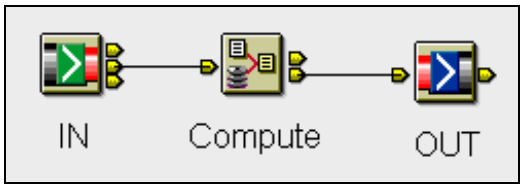
Although these accommodate the functions needed to handle the majority of data formats, there may still be opportunities for additional parsers to handle additional generic or specific formats.

This SupportPac documents and provides just such an additional parser called **TREEASIS** that accommodates new forms of generic data.

Chapter 2. Background to Parsers

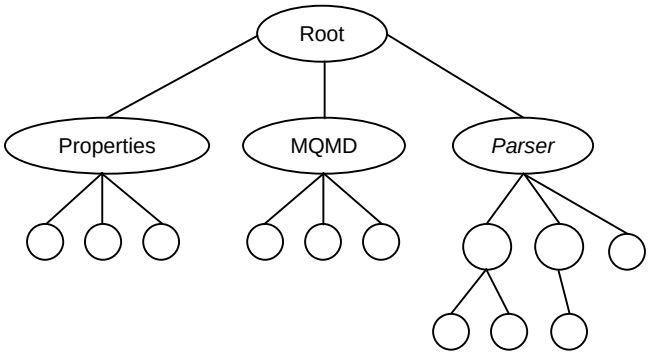
Before explaining the details of the TREEASIS parser, it is highly beneficial to have a good understanding of how MQSI V2 parsers operate.

In the following trivial message flow, both MQInput (IN) and MQOutput (OUT) nodes are illustrated. These nodes are associated with MQSeries queues. When a message arrives on the queue associated with the MQInput node, the node will get the message from the queue and parse its contents.



The parser operates by interpreting the contents of the binary data retrieved from the queue and building a *message tree* from that data. The design and implementation of the parser dictates how the message tree should be built.

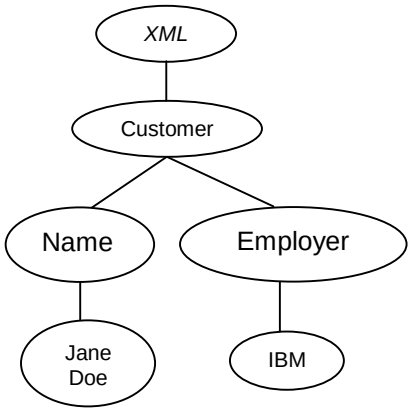
A full message tree has the following generic format:



The message tree is a logical representation of the original incoming binary data in the incoming MQSeries message. The Properties and MQMD sub trees are automatically added by MQSI for each message. An MQSI parser creates the right-most sub tree illustrated above. The name of the node immediately beneath the Root node is that of the parser. Commonly this will be seen as one of XML, MRM, NEON or BLOB.

The architecture of MQSI allows additional parsers to be designed, implemented and made available. These custom parsers can take the incoming message data and build an associated message tree.

When a message reaches an MQOutput node, a parser will be re-invoked. The purpose at this time will be to take the message tree currently in effect and produce a binary representation of the tree. This data will then be written as a message to a queue. The "un-parsing" or re-combination of the elements of the tree is controlled by the logic of the parser. For example, a tree with the following structure:



Will produce a message with binary format of:

```
<Customer>
<Name>Jane Doe</Name>
<Employer>IBM</Employer>
</Customer>
```

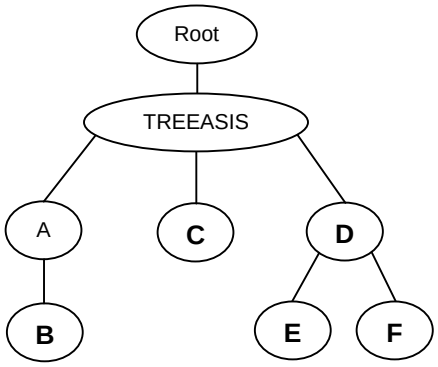
As can be seen, a parser usually adds additional data format to that of the message tree prior to writing to the queue. For XML, the parser adds angle brackets to make a well-formed XML data stream. For other parsers, for example MRM and NEON, the actual format of the output data is a combination of both the message tree and a description of the format to use stored elsewhere. For MRM, this format is stored in the message repository database and for NEON, the format is stored in a database by the NEON Formatter.

Chapter 3. The TREEASIS Parser

Compute nodes, Extract nodes and other such nodes perform the manipulation of a message tree within a message flow. The coding of the business logic is performed in the ESQL language; the elements of the message tree can be added, deleted, copied or modified. The programmers who design the message flows are usually accomplished in thinking in terms of the message tree and the effects of ESQL statements on that tree. When the message tree reaches an MQOutput node, the tree structure is lost and the data reformatted to a message on a queue.

The TREEASIS parser is intrinsically very simple. It *walks* the message tree and, for each node in the tree that contains a value, writes its binary representation as part of the outgoing message.

For example, in the following tree:



The nodes marked BOLD have values. When the TREEASIS parser navigates the tree, it will create an output data stream containing the values:

```
B C D E F
```

The benefits and use of this can be quickly seen when, for example, contemplating variable or delimited data.

Example: Colon delimited, variable length data

An input piece of data consists of XML in the following form:

```
<Customer>
<Name>Jane Doe</Name>
<Street>123 Elm</Street>
<City>Anytown</City>
<State>TX</State>
</Customer>
```

It is desired to transform this data to a colon-delimited format:

```
Jane Doe:123 Elm:Anytown:TX:
```

Using the MQSI supplied parsers, such a transformation would be very difficult. Using the TREEASIS parser, such a transformation becomes much easier:

```
DECLARE I INTEGER;
SET I = 1;
WHILE I < CARDINALITY(InputRoot.*[]) DO
       SET OutputRoot.*[I] = InputRoot.*[I];
       SET I=I+1;
END WHILE;

-- Enter SQL below this line.  SQL above this line might be regenerated, causing any
modifications to be lost.

SET "OutputRoot"."TREEASIS"."Value" = "InputBody"."Customer"."Name" || ':';
SET "OutputRoot"."TREEASIS"."Value" = "OutputRoot"."TREEASIS"."Value" ||
"InputBody"."Customer"."Street" || ':';
SET "OutputRoot"."TREEASIS"."Value" = "OutputRoot"."TREEASIS"."Value" ||
"InputBody"."Customer"."City" || ':';
SET "OutputRoot"."TREEASIS"."Value" = "OutputRoot"."TREEASIS"."Value" ||
"InputBody"."Customer"."State" || ':';
```

Only the values and order of the elements in the message tree are important when a message is processed for output by the TREEASIS parser. The names of the fields are unimportant. A functionally identical alternative to the previous ESQL is illustrated next:

```
DECLARE I INTEGER;
SET I = 1;
WHILE I < CARDINALITY(InputRoot.*[]) DO
       SET OutputRoot.*[I] = InputRoot.*[I];
       SET I=I+1;
END WHILE;
-- Enter SQL below this line.  SQL above this line might be regenerated, causing any
modifications to be lost.
SET "OutputRoot"."TREEASIS"."Name" = "InputBody"."Customer"."Name" || ':';
SET "OutputRoot"."TREEASIS"."Street" = "InputBody"."Customer"."Street" || ':';
SET "OutputRoot"."TREEASIS"."City" = "InputBody"."Customer"."City" || ':';
SET "OutputRoot"."TREEASIS"."State" = "InputBody"."Customer"."State" || ':';
```

The following would also produce identical output:

```
DECLARE I INTEGER;
SET I = 1;
WHILE I < CARDINALITY(InputRoot.*[]) DO
       SET OutputRoot.*[I] = InputRoot.*[I];
       SET I=I+1;
END WHILE;
-- Enter SQL below this line.  SQL above this line might be regenerated, causing any
modifications to be lost.
SET "OutputRoot"."TREEASIS"."A" = "InputBody"."Customer"."Name" || ':';
SET "OutputRoot"."TREEASIS"."B" = "InputBody"."Customer"."Street" || ':';
SET "OutputRoot"."TREEASIS"."C" = "InputBody"."Customer"."City" || ':';
SET "OutputRoot"."TREEASIS"."D" = "InputBody"."Customer"."State" || ':';
```

MQSI knows to utilize the TREEASIS parser to produce the output message because that is the name of the tree node immediately subordinate to Root.

Unlike other parsers, the TREEASIS parser may **only** be utilized to generate output data. It has no meaning if there is an attempt to parse incoming data in an MQInput node using TREEASIS to format the data. If attempted, the parser will detect this invalid use and throw a catchable exception.

Installation of the TREEASIS parser

The TREEASIS parser is supplied as an MQSI Loadable Implementation Library (LIL). On NT, this is a DLL and the file should be copied to:

C:\Program Files\IBM MQSeries Integrator 2.0.1\bin

On AIX, the LIL file should be copied to:

/usr/opt/mqsi/lil

and the messages file (MQSIV2_TREEASIS.cat) copied to:

/usr/opt/mqsi/messages

Data types supported

The TREEASIS parser supports message tree content with the following data types:

Character

The character string is the most common type of data. When a string value is contained in a TREEASIS node in the message tree, it is output as ASCII (code page 1208). For example, the following statement:

SET "OutputRoot"."TREEASIS"."A" = 'ABCD1234';

will result in the 8 characters "ABCD1234" in the final output stream.

Byte array (BLOB)

The byte array is an un-interpreted array of bytes. For example, the following statement:

```
SET "OutputRoot"."TREEASIS"."C" = CAST('F0E1D2C3' AS BLOB);
SET "OutputRoot"."TREEASIS"."D" = "InputRoot"."MQMD"."MsgId";
```

will result in 4 bytes of data (0x'F0E1D2C3') followed by the MsgId from the MQMD structure.

Integer

An integer value is output as a machine native 64-bit representation of the value.

Additional Information

Additional information on MQSI parsers may be found in the MQSeries Integrator Programming Guide (SC34-5603) for Chapter 6 onwards.

Future enhancements

At this time, future enhancements will be added on a best-effort basis. Candidates for additional functions include:

- Support for additional data types.
- The ability to specify the size of data types (e.g. 32-bit integer)
- Build for Solaris (based upon demand for this function)

Support

Support for this parser including defect reports and enhancement requests will be supplied on a best effort basis.

Please send any support requests by email to:

Attn: Neil Kolban

kolban@us.ibm.com

Please include as much detail as possible and, a "Trace" node output of the message tree immediately prior to the failing node.