XML File Access Guide

Release 8.1.3

November 2013

IKAN Solutions N.V.
Kardinaal Mercierplein 2
B-2800 Mechelen
BELGIUM

Table of Contents

CHAPTER 1

About This Manual

MetaSuite File Access for XML is intended for users with some experience with MetaSuite. It provides the information you need to use the MetaSuite XML Access.

1.1. Prerequisites

Readers are expected to be familiar with XML.

1.2. Related Publications

The MetaSuite User and Reference Guides describe the different MetaSuite components and provide examples for using MetaSuite. Those guides should be available for reference during the installation and test procedures described here.

The following table gives an overview of the complete MetaSuite documentation set.

Release Information	Release Notes 8.1.3
Installation Guides	• BS2000/OSD Runtime Component • DOS/VSE Runtime Component • Fujitsu Windows Runtime Component • MicroFocus Windows Runtime Component • MicroFocus UNIX Runtime Component • OS/390 and Z/OS Runtime Component • OS/400 Runtime Component • VisualAge Windows Runtime Component • VisualAge UNIX Runtime Component • VMS Runtime Component
User Guides	• INI Manager User Guide • Installation and Setup Guide • Introduction Guide • MetaStore Manager User Guide • MetaMap Manager User Guide • Generator Manager User Guide
Technical Guides	• ADABAS File Access Guide • IDMS File Access Guide • IMS DLI File Access Guide • RDBMS File Access Guide • XML File Access Guide • Runtime Modules • User-defined Functions User Guide

If you are unfamiliar with MetaSuite, the following technical description provides you with a brief overview.

The MetaSuite System	MetaSuite is designed for data retrieval, extraction, conversion and reporting. It includes a workstation-based graphical user interface and a mainframe runtime component.
MetaSuite Database Interfaces	MetaSuite can access data from a number of database management systems, using the same commands, program structure and retrieval techniques used for non-database files. Each database interface is available as an optional enhancement to the base product.
MetaMap Manager	MetaMap Manager is the MetaSuite tool used to define models. Such models are intuitively built by describing overall program specifications, input file definitions (data and process) and target file definitions (data and process).
MetaStore Manager	MetaStore Manager is a tool that provides metadata maintenance and documentation services.
Generator Manager	The Generator Manager is the system administration tool.All kinds of basic functionalities and customization possibilities are supported by this tool.

CHAPTER 2

Reading XML source files

2.1. Overview

This section describes which XML features are supported by MetaSuite in case of reading an XML file.

2.2. Basics

The simplest case of an XML structure is the structure generated by the generator for XML target files.

```
[ <recordname> [ <fieldname> fieldcontent </fieldname> ] </recordname> ]
```

The record name in the record node must correspond to the record name defined in the MetaStore dictionary.

2.3. Group fields

A first extension of this syntax is the definition of group fields and sub fields.
Field content can be described as:

```
fieldcontent = <subfieldname> fieldcontent </subfieldname> | value
```

So, a field can contain a sub field, which might again contain another sub field.
Example:

```
<record1>
  <person>
    <name>
      <forename> "Filip" </forename>
      <familyname> "Bastien" </familyname>
    </name>
    <address>
      <street> "Hoogland" </street>
      <community> "Werchter" </community>
    </address>
  </person>
  <animal> "monkey" </animal>
</record1>
```

2.4. Attributes

An xml-node can have attributes.

```
<node-name [attribute-name = "attribute-value" ] >
```

Those attributes will be treated equally as fields.

Example:

In this sample the sub fields from previous example were filled in as attributes.

For MetaSuite there is no difference at all between this and previous example.

```
<record1>
  <person>
    <name forename= "filip" familyname="bastien">
    </name>
    <address street="hoogland" community="Werchter">
    </address>
  </person>
</record1>
```

- Node names and attribute names are internally uppercased, in other words, there is no difference between lowercase and uppercase nodes or attributes.
- Neither node values nor attribute values will be uppercased.
- Attribute values must be surrounded by single or double quotes, even when the content is numeric. MetaSuite knows what to do with it.
- In order to include quotes in the attribute values, just double them.
- Ex: 'ABC''DEFG' will become ABC'DEFG in memory.
- XML attributes will be collected with the property XMLTYPE = 'ATTRIBUTE'.

2.5. Simple or empty nodes

An XML-node can be defined without an end node, just by putting an ending slash before the end of the node.

```
<node content= "x"> </node>
```

is the same as:

```
<node content="x"/>
```

This is the so called "empty node", because no content follows this node.

Our example from above can be rewritten as follows:

```
<record1>
  <person>
    <name forename= "filip" familyname="bastien"/>
    <address street="hoogland" community="Werchter"/>
  </person>
</record1>
```

2.6. Not defined fields

Attributes and/or nodes that aren't defined in the record description in the MetaStore will not be read.

This is the same as for other MetaSuite source fields: you don't have to define them all, as you don't need to use them all.

2.7. Multiple occurrences

Fields that are defined multiple times within one record may be defined as arrays in the MDL definition.

Example:

```
<authorgroup>
  <author>
    <firstname>Michel</firstname>
    <surname>Goossens</surname>
  </author>
  <author>
    <firstname>Frank</firstname>
    <surname>Mittelbach</surname>
  </author>
  <author>
    <firstname>Alexander</firstname>
    <surname>Samarin</surname>
  </author>
</authorgroup>
```

If authorgroup is the record name, then "firstname" and "surname" can be defined as arrays. However, if you define "author" as the record name, then the node "<authorgroup>" will be ignored and per "author" record there will only be one occurrence of "firstname" and "surname" per record.

We will assume that you want to treat "authorgroup" as record.

If "firstname" is defined as a multiple occurring field within the record, then there is no problem. "Michel" will be put in the field "firstname (1)", "Frank" will become "firstname (2)" etc…

If "firstname" is NOT defined as a multiple occurring field within the record, then there is again no problem. "Alexander" will be put in the field "firstname", because the last value of "firstname" will override the previous values.

2.8. XML entities

XML has the possibility to define parameters.

Example:

```
<!ENTITY FIB "Filip Bastien">
```

Every occurrence of '`&FIB;`' will be replaced by "Filip Bastien"… but not in MetaSuite.

This XML feature is not supported yet!

Pre-programmed entities like '`&nspb;`' are not supported either.

You may however include nodes containing entity definitions and entity values, but MetaSuite will ignore the definitions and will not translate the entity values.

2.9. XML processing instructions

XML gives us the possibility to define parameters that may be passed to the main program.
Example:

```
<?xml version="1.0"?>
<?runinfo DEBUG="YES"?>
```

The XML parser, MetaSuite in this case, might interpret such information as "DEBUG=YES".
But… at this time we don't support this feature.
You may however include nodes containing processing instructions.

2.10. XML comment

XML comment lines.
Example:

```
<! Ceci n'est pas une pipe ! is comment 7>5<6>
```

This feature is not supported yet, but you may include comment lines if you avoid using delimiter characters like "<", ">" and quotes.

2.11. CDATA

In order to use special characters and multiple spaces, CDATA is often used.
CDATA sample:

```
<![CDATA[here you can </b> write <i> all that you want <p> and the
XML parser won't complain]]>¶
```

Line breaks will not be taken in account. They will be replaced by a space.

2.12. Validation rules

The MetaSuite Generator validates the "well-ness" of the XML syntax and MetaStore validates the XSD, but there is no hard validation of the XSD.
However, field validation occurs to a certain point:

- XSD fields will be translated into character fields, numeric fields, dates.
- When reading fields that are supposed to be numeric or a date in a certain format, the MetaSuite transformation program checks the field content and will exclude bad records.

2.13. MDL definition

The MDL definition of XML records reflects the way how XML records will be stored in memory.
Example:

```
ADD FILE XMLSAMP2 TYPE XML VARIABLE 251 BLOCK 251 LABEL STANDARD
ADD RECORD invoice SIZE 20
ADD FIELD invoice_id POSITION 1 SIZE 5 TYPE ZONED UNSIGNED
ADD FIELD client_id POSITION 6 SIZE 5 TYPE ZONED
ADD FIELD curr_date POSITION 11 SIZE 10 TYPE CHARACTER DATE 'DD?MM?YYYY'
ADD RECORD return SIZE 20
ADD FIELD return_id POSITION 1 SIZE 5 TYPE ZONED UNSIGNED
ADD FIELD client_id POSITION 6 SIZE 5 TYPE ZONED
ADD FIELD curr_date POSITION 11 SIZE 10 TYPE CHARACTER DATE 'DD?MM?YYYY'
```

The XML input file may look like this:

```
<?XML version="1.0"?>
<invoice>
  <invoice_id> 3125 </invoice_id>
  <client_id> 25956 </client_id>
  <curr_date> 30/10/2002 </curr_date>
  <order>
    <product>
      <product_id> B417-1 </product_id>
      <quantity> 154 </quantity>
    </product>
    <product>
      ... etc ...
    </product>
  </order>
  <payment>
    <mode> banksys </mode>
    <credit_card> 001-4551111-11 </credit_card>
    <amount> 5514.15 </amount>
  </payment>
</invoice>
<return>
  <invoice_id> 3125 </invoice_id>
  <client_id> 25956 </client_id>
  <curr_date> 31/10/2002 </curr_date>
  <returnparts>
    <product>
      <product_id> B417-1 </product_id>
      <quantity> 44 </quantity>
    </product>
  </returnparts>
  <return_payment>
    <mode> cash </mode>
    <credit_card> 001-4551111-11 </credit_card>
    <amount> 200.70 </amount>
  </return_payment>
</return>
<invoice>
<invoice_id> 5484 </invoice_id>
...etc...
```

Nodes as well as attributes can be defined as record fields – this gives users a great liberty of defining input records and their structure. The file above can be defined as a collection of "invoice" records, but also as a collection of "payment" records.

Another correct XML input file that corresponds with the given MDL might look like this:

```
<?XML version="1.0"?>
<invoice invoice_id="3125" client_id="25956" curr_date="30/10/2002" />
<return invoice_id="3126" client_id="25956" curr_date="31/10/2002" />
...etc...
```

And again, every field that you don't define in de MetaStore dictionary will not be read in.

2.14. Null

Nullable fields (INNULL, OUTNULL or OUTNULR) will get the null value in two cases:

- The corresponding node or attribute is not found in the record
- The corresponding node contains no value.

2.15. Code-control XML or XPC

The first version of XML for MetaSuite (version 7.2.1.) did not check the path. Code-control table XMLINP was used.

XML files often have equal node names or attribute names with a different meaning. Equal names occur in a different context. In this case the path name has to be checked before moving a value to a certain variable. The current version takes care of this inconvenient situation. The XML path structure corresponds to the MDL group structure.

Another code-control table is used in that case: The code-control table XPC is used. XPC is the abbreviation of XML with PATH CONTROL.

Example:

Consider following XML file:

```
<XML>
  <FAMILY>
    <FATHER name="PETER" surname="PAN"/>
    <MOTHER name="AXELLE" surname="RED"/>
    <CHILD name="DAVID" surname="PAN"/>
    <CHILD name="LINDA" surname="PAN"/>
  </FAMILY>
</XML>
```

FATHER, MOTHER and CHILD contain both "name" and "surname". In version 7.2.1. XML support, the difference between those attributes could not be made. In version 7.2.2. the user can use the XPC code-control in order to distinguish the surname of FATHER, MOTHER and CHILD.

2.16. Multi Record Sources - Basic XML Parsing Without XPATH

An example often tells more than a thousand words, so we start with one:

```
ADD FILE XMLSAMP2 TYPE XML VARIABLE 251 BLOCK 251
ADD RECORD faktuur#08757 SIZE 20
ADD FIELD faktuurid#08758 POSITION 1 SIZE 5 TYPE ZONED UNSIGNED
ADD FIELD klantid#08759 POSITION 6 SIZE 5 TYPE ZONED
ADD FIELD datum#08760 POSITION 11 SIZE 10 TYPE CHARACTER DATE 'DD?MM?YYYY'
ADD RECORD retour#08761 SIZE 20
ADD FIELD faktuurid#08762 POSITION 1 SIZE 5 TYPE ZONED UNSIGNED
ADD FIELD klantid#08763 POSITION 6 SIZE 5 TYPE ZONED
ADD FIELD datum#08764 POSITION 11 SIZE 10 TYPE CHARACTER DATE 'DD?MM?YYYY'
```

The XML parser will search for <faktuur> nodes and for <retour> nodes.

No key definition is needed! The distinction between two record types is made by the node name.

In order to separate the different record types you can test on SYS-RECORD.

Example:

```
BEGIN TARGETFILE 1 INPUT
CASE SYS-RECORD -
EQ '----faktuur' -
  PUT ( 1 )
BEGIN TARGETFILE 2 INPUT
CASE SYS-RECORD -
EQ '----retour' -
  PUT ( 2 )
```

In fact, there is not much difference between XML target files and other target files.

2.17. Multi Record Sources - Parsing using the XPATH property

XPATH is a query language for selecting nodes from an XML document. The XPATH contains the concatenation of the different nodes that have to be read before the actual content starts.

The XPATH delimiter is a forward slash.

Example:

```
ADD FILE COLLECT_TYPES TYPE XML CODE-CONTROL XPC XPATH '/XML/Profiles'
XML-DECLARATION ...
```

No record verification will be done until the nodes <XML> and <Profiles> have been read.

When reading a source file, the full XPATH will be read before the record parsing is started.

The path of the 'record' node can be put in the XPATH property of the file and/or record description. If both of them are available and the XPATH of the record is not an absolute XPATH (meaning that the XPATH does not start with a forward slash), the XPATH properties of file and record will be merged.

If the XPATH of the record is absolute (starts with a forward slash), then the XPATH of the file will not be taken in account.

Example 1

```
ADD FILE FAMILYFILE TYPE XML VARIABLE 54 BLOCK 54 XPATH '/XML/FAM'
ADD RECORD FAMILY SIZE 54 XPATH '/XML/FAM/FAMILY'
```

In this sample the XPATH of the record is an absolute path, and will not be concatenated with the XPATH of the file.

Example 2

```
ADD FILE FAMILYFILE TYPE XML VARIABLE 54 BLOCK 54 XPATH '/XML'
ADD RECORD FAMILY SIZE 54 XPATH 'FAM'
```

Here the XPATH of the record is a relative path, and will be concatenated with the XPATH of the file. The XPATH of the record will become /XML/FAM.

2.18. TAGVALUE

When a node contains subnodes and/or attributes, this node corresponds to a group structure. For MetaSuite (the MGL is COBOL) a group can not have a value on its own. The MetaSuite group value is the value of all sub-elements combined.

XML has the possibility to define nodes that have attributes and subnodes, and a value of their own. MetaSuite uses the reserved word TAGVALUE to solve this double meaning.
If node x can have a value of its own, please define the field TAGVALUE as a subfield of field x in the MDL definition.

2.19. XML-NAME

The XML-NAME is a field property that is used for linking an XML field with the name of a node. If this property is missing, the field name is used as default value for this property.

2.20. XSD properties and their translation into MDL properties

- maxOccurs="unbounded"

  This property value can be interpreted by the User as an array with an infinite number of records.

  In order to save storage, and to make it possible to handle this kind of features, the MetaStore Manager collects this structure as a separate record.
- minOccurs="0" and maxOccurs not defined or "1"

  Will be collected as a nullable value (INNULL)
- maxOccurs="2" or higher

  Will be collected as an Array.
- type = "xs:string"

  will be collected as character. The default size can be set within the INI manager.
- type = "xs:decimal"

  will be collected as numeric. The default size can be set within the INI manager.
- type = "xs:date"

  will be collected as an ISO character date, with date mask "YYYY?MM?DD"
- xs:minExclusive value="0" / xs:minInclusive value="0"

  will be considered as being UNSIGNED.
- xs:fractionDigits value="2"

  The fractionDigits will be colleted as the number of decimals after the decimal point, i.e., the DECIMALS value.
- xs:maxLength value="7"

  The maxLength property will overrule the default size that is attributed to a certain xs-type.

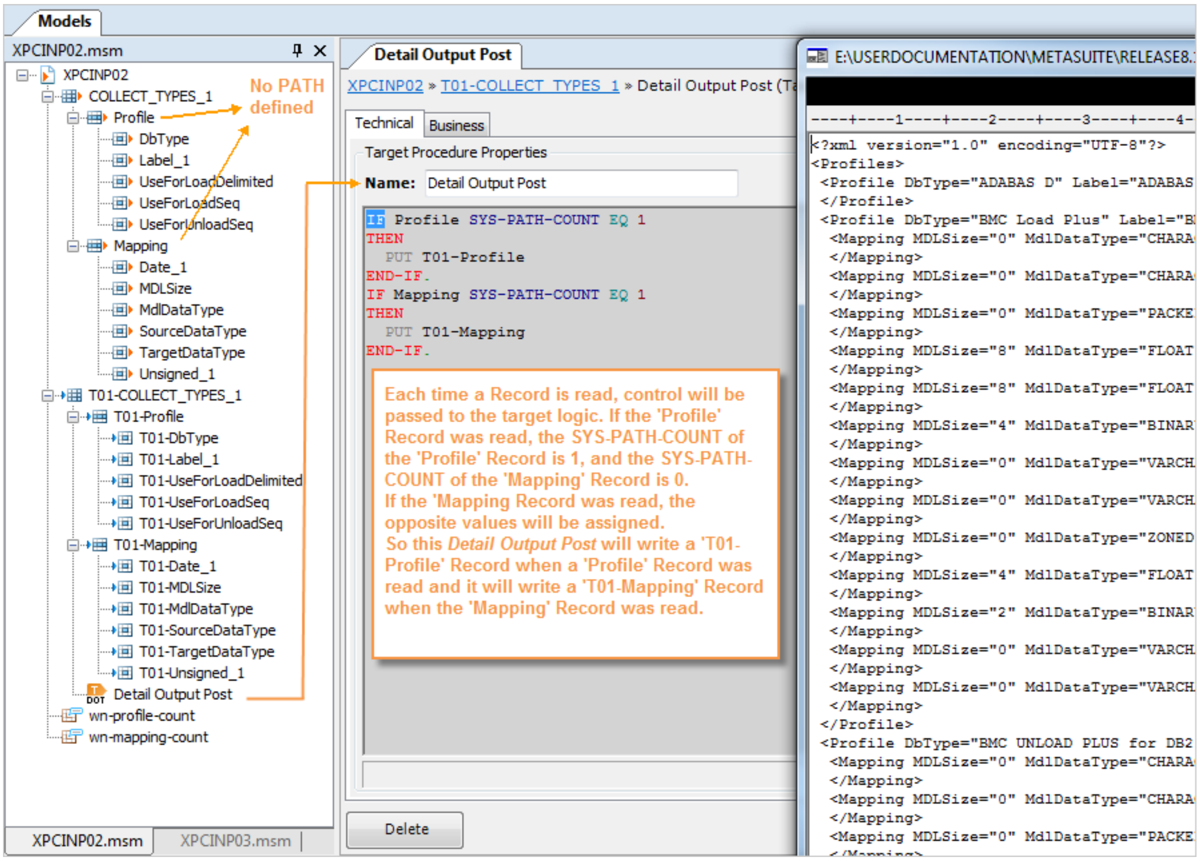
2.21. PATH

Adding a path to the source file changes the read/write sequence. Multiple reads can be done before starting the write sequence.

Without adding the path, the property SYS-PATH-COUNT of the record that has been read will have the value ONE. The SYS-PATH-COUNTs of the other records will be ZERO.

When adding a path to the file, the property SYS-PATH-COUNT of the records that have been read will have value ONE or more (in case of multiple occurring paths).

Example without path:



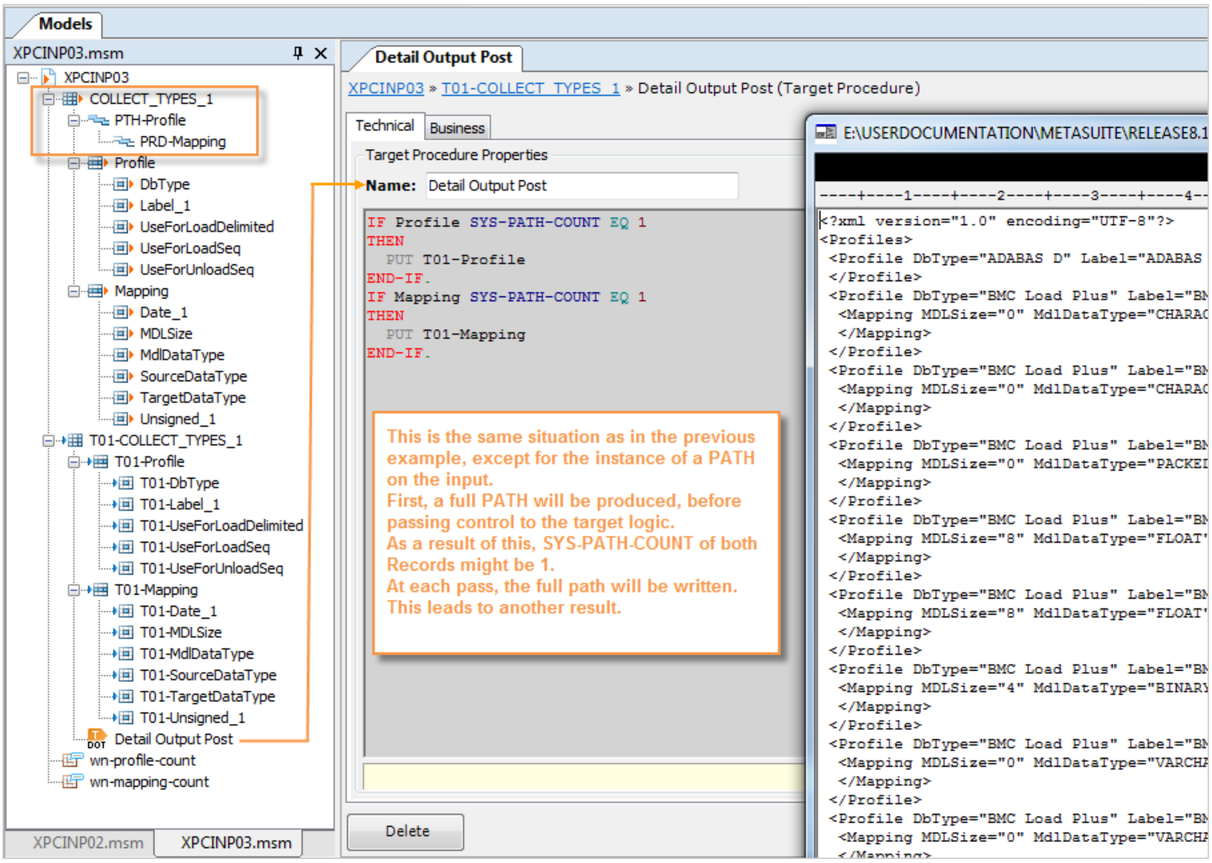
Name: Detail Output Post

```
IF Profile SYS-PATH-COUNT EQ 1
THEN
  PUT T01-Profile
END-IF.
IF Mapping SYS-PATH-COUNT EQ 1
THEN
  PUT T01-Mapping
END-IF.
```

Each time a Record is read, control will be passed to the target logic. If the 'Profile' Record was read, the SYS-PATH-COUNT of the 'Profile' Record is 1, and the SYS-PATH-COUNT of the 'Mapping' Record is 0.
If the 'Mapping Record was read, the opposite values will be assigned.
So this *Detail Output Post* will write a 'T01-Profile' Record when a 'Profile' Record was read and it will write a 'T01-Mapping' Record when the 'Mapping' Record was read.

```
----+----1----+----2----+----3----+----4-
<?xml version="1.0" encoding="UTF-8"?>
<Profiles>
 <Profile DbType="ADABAS D" Label="ADABAS
 </Profile>
 <Profile DbType="BMC Load Plus" Label="B
  <Mapping MDLSize="0" MdlDataType="CHARA
  </Mapping>
  <Mapping MDLSize="0" MdlDataType="CHARA
  </Mapping>
  <Mapping MDLSize="0" MdlDataType="PACKE
  </Mapping>
  <Mapping MDLSize="8" MdlDataType="FLOAT
  </Mapping>
  <Mapping MDLSize="8" MdlDataType="FLOAT
  </Mapping>
  <Mapping MDLSize="4" MdlDataType="BINAR
  </Mapping>
  <Mapping MDLSize="0" MdlDataType="VARCH
  </Mapping>
  <Mapping MDLSize="0" MdlDataType="VARCH
  </Mapping>
  <Mapping MDLSize="0" MdlDataType="ZONED
  </Mapping>
  <Mapping MDLSize="4" MdlDataType="FLOAT
  </Mapping>
  <Mapping MDLSize="2" MdlDataType="BINAR
  </Mapping>
  <Mapping MDLSize="0" MdlDataType="VARCH
  </Mapping>
  <Mapping MDLSize="0" MdlDataType="VARCH
  </Mapping>
 </Profile>
 <Profile DbType="BMC UNLOAD PLUS for DB2
  <Mapping MDLSize="0" MdlDataType="CHARA
  </Mapping>
  <Mapping MDLSize="0" MdlDataType="CHARA
  </Mapping>
  <Mapping MDLSize="0" MdlDataType="PACKE
```

Example with path:



Name: Detail Output Post

```
IF Profile SYS-PATH-COUNT EQ 1
THEN
  PUT T01-Profile
END-IF.
IF Mapping SYS-PATH-COUNT EQ 1
THEN
  PUT T01-Mapping
END-IF.
```

This is the same situation as in the previous example, except for the instance of a PATH on the input.
First, a full PATH will be produced, before passing control to the target logic.
As a result of this, SYS-PATH-COUNT of both Records might be 1.
At each pass, the full path will be written.
This leads to another result.

```
----+----1----+----2----+----3----+----4--
<?xml version="1.0" encoding="UTF-8"?>
<Profiles>
 <Profile DbType="ADABAS D" Label="ADABAS
 </Profile>
 <Profile DbType="BMC Load Plus" Label="BM
  <Mapping MDLSize="0" MdlDataType="CHARAC
  </Mapping>
 </Profile>
 <Profile DbType="BMC Load Plus" Label="BM
  <Mapping MDLSize="0" MdlDataType="CHARAC
  </Mapping>
 </Profile>
 <Profile DbType="BMC Load Plus" Label="BM
  <Mapping MDLSize="0" MdlDataType="PACKED
  </Mapping>
 </Profile>
 <Profile DbType="BMC Load Plus" Label="BM
  <Mapping MDLSize="8" MdlDataType="FLOAT"
  </Mapping>
 </Profile>
 <Profile DbType="BMC Load Plus" Label="BM
  <Mapping MDLSize="8" MdlDataType="FLOAT"
  </Mapping>
 </Profile>
 <Profile DbType="BMC Load Plus" Label="BM
  <Mapping MDLSize="4" MdlDataType="BINARY
  </Mapping>
 </Profile>
 <Profile DbType="BMC Load Plus" Label="BM
  <Mapping MDLSize="0" MdlDataType="VARCHA
  </Mapping>
 </Profile>
 <Profile DbType="BMC Load Plus" Label="BM
  <Mapping MDLSize="0" MdlDataType="VARCHA
  </Mapping>
```

CHAPTER 3

Writing XML target files

3.1. Overview

This section describes which XML features are supported by MetaSuite.

3.2. Basics

The simplest case of an XML structure is the structure generated by the generator for XML target files.

```
[ <recordname> [ <fieldname> fieldcontent </fieldname> ] </recordname> ]
```

This is the record structure that will be maintained if XML target files are written.

3.3. XML-DECLARATION

This feature is used for target XML files, and is not needed for source files.

It contains the XML declaration sentence that is written as leading part of the XML file.

3.4. XML-NAME

The XML-NAME is a field property that is used for linking an XML field with the name of a node. If this property is missing, the field name is used as default value for this property.

3.5. XML-TYPE

This field property can have the value "ATTRIBUTE", "GROUP" or "NODE". Only the value "ATTRIBUTE" is important.

If the XML-TYPE of a field is an attribute, this field will be added as an attribute to the group field or record to which it belongs.

Attributes can be attributed to XML (group)fields and XML records, but not to XML files.

Therefore, if the user wants to have an attribute on an XML record structure, this XML structure must have a record path! It does not have to be an absolute path, but it can not be empty in case of attributes on a record.

3.6. Simple MDL definition

If one wants to define an XML record, define the definition of how it will be stored in memory, because a fixed layout of an XML record does not exist.

Example:

```
ADD FILE XMLSAMPO TYPE XML VARIABLE 251 BLOCK 251 LABEL STANDARD
ADD RECORD invoice SIZE 20
ADD FIELD invoice_id POSITION 1 SIZE 5 TYPE ZONED UNSIGNED
ADD FIELD client_id POSITION 6 SIZE 5 TYPE ZONED
ADD FIELD curr_date POSITION 11 SIZE 10 TYPE CHARACTER DATE 'DD?MM?YYYY'
ADD RECORD return SIZE 20
ADD FIELD return_id POSITION 1 SIZE 5 TYPE ZONED UNSIGNED
ADD FIELD client_id POSITION 6 SIZE 5 TYPE ZONED
ADD FIELD curr_date POSITION 11 SIZE 10 TYPE CHARACTER DATE 'DD?MM?YYYY'
```

The target definition is almost the same as that of a delimited output file:

```
REMARKS datawarehouse (XML)
TARGETFILE 1 DELIMITED OUTPUT-CONTROL XML
TITLE 0 ('T01-datawarehouse')
DETAIL 1 RECORD 'PP_employee' -
( -
 T01-employee_number, -
 T01-employee_name, -
 T01-annual_salary, -
 T01-bonus, -
 T01-pay_code, -
 T01-date_of_hire, -
 T01-date_time, -
 T01-time_to_work -
)
```

The main difference is the OUTPUT-CONTROL parameter, which must be XML, and the RECORD parameter, which contains the record name.

If this parameter is omitted, then the -name will become targetfile-xx-detail-yy. The output file will look something like:

```
<XML>
  <PP_employee>
    <employee_number>0</employee_number>
    <employee_name>" "</employee_name>
    <annual_salary>.00</annual_salary>
    <bonus>20.00</bonus>
    <pay_code>" "</pay_code>
    <date_of_hire>00000000</date_of_hire>
    <date_time>" "</date_time>
    <time_to_work>" "</time_to_work>
  </PP_employee>
  <PP_employee>
    <employee_number>0</employee_number>
    <employee_name>" "</employee_name>
    <annual_salary>.00</annual_salary>
    <bonus>20.00</bonus>
    <pay_code>" "</pay_code>
    <date_of_hire>00000000</date_of_hire>
    <date_time>" "</date_time>
    <time_to_work>" "</time_to_work>
  </PP_employee>
  <PP_employee>
    <employee_number>0</employee_number>
    <employee_name>" "</employee_name>
    <annual_salary>.00</annual_salary>
    <bonus>20.00</bonus>
    <pay_code>" "</pay_code>
    <date_of_hire>00000000</date_of_hire>
    <date_time>" "</date_time>
    <time_to_work>" "</time_to_work>
  </PP_employee>
</XML>
```

3.7. Enhanced MDL definition

More complicated MDL definitions have code control property "XPC".

While the output of the simple MDL is a flat structure, the outcome of an enhanced MDL definition follows the rules of the XPATH, XML-NAME, XML-TYPE and XML-DECLARATION.

A flat structure is not needed. The structure of the record is reflected in the structure of the XML target file.

Multi-record structures can be written by using PUT statements.

3.8. Null fields

Null fields can be omitted by setting MTL option DROP-XML-NULL-FIELD

This option can be set to the following values:

- "ON" : Null fields will be dropped.
- "OFF" : Null fields will be tagged.

3.9. XPATH

XPATH is a query language for selecting nodes from an XML document. The XPATH contains the concatenation of the different nodes that have to be read before the actual content starts.

The XPATH delimiter is a forward slash.

Example:

```
ADD FILE COLLECT_TYPES TYPE XML CODE-CONTROL XPC XPATH '/XML/Profiles'
XML-DECLARATION ...
```

When writing a target file, the full XPATH will be written before the record node is put.

The path of the record can be put in the XPATH property of the file and/or record description. If both of them are available and the XPATH of the record is not an absolute XPATH, the XPATH properties of file and record will be merged.

If the XPATH of the record is absolute (starts with a forward slash), then the XPATH of the file will not be taken in account.

Example 1

```
ADD FILE FAMILYFILE TYPE XML VARIABLE 54 BLOCK 54 XPATH '/XML/FAM'
ADD RECORD FAMILY SIZE 54 XPATH '/XML/FAM/FAMILY'
```

In this sample the XPATH of the record is an absolute path, and will not be concatenated with the XPATH of the file.

Example 2

```
ADD FILE FAMILYFILE TYPE XML VARIABLE 54 BLOCK 54 XPATH '/XML'
ADD RECORD FAMILY SIZE 54 XPATH 'FAM'
```

Here the XPATH of the record is a relative path, and will be concatenated with the XPATH of the file. The XPATH of the record will become /XML/FAM.

3.10. Write-Behind

In the programming world, the principle of READ-AHEAD is known. The central idea of the "read ahead" is that, because the end of the file cannot be detected until an attempt is made to read a record, the Read must be positioned as the last statement in the record processing loop.

For XML files, when writing a node, we use the opposite technique: Write-Behind. MetaSuite does not close a node until it knows which node has to be put next. If the new node to be put is a subnode, the closing will be delayed.

At the end of the target processing, all open nodes will be closed.

APPENDIX A

Data Administration

A.1. Collecting XSD Files

Property	Description
maxOccurs="unbounded"	This property value can be interpreted by the user as an array with an infinite number of records. In order to save storage, and to make it possible to handle this kind of features, the MetaStore Manager collects this structure as a separate record.
minOccurs="0" and maxOccurs not defined or "1"	Will be collected as a nullable value (INNULL), also if IsNullable attribute is set.
maxOccurs="2" or higher	Will be collected as an array.
type = "xs:string"	Will be collected as character. The default size can be set within the INI manager.
type = "xs:decimal"	will be collected as numeric. The default size can be set within the INI manager.
type = "xs:date"	Will be collected as an ISO character date, with date mask "YYYY?MM?DD".
MinExclusive/MinInclusive	The MinExclusive/MinInclusive value goes into the LOWLIMIT of the field
MaxExclusive/MaxInclusive	The MaxExclusive/MaxInclusive is put into the HIGHLIMIT of the field.
xs:minExclusive value="0" / xs:minInclusive value="0"	Will be considered as being UNSIGNED.
xs:fractionDigits value="2"	The fractionDigits will be collected as the number of decimals behing the decimal point, i.e. the DECIMALS value.
xs:maxLength value="7"	The maxLength property will overrule the default size that is attribuated to a certain xs-type.
fixed="FIXEDVALUE"	Will be translated into Initial value = 'FIXEDVALUE' , Low Limit = 'FIXEDVALUE' and High Limit = 'FIXEDVALUE' .
TotalDigits	TotalDigits will be put in the SIZE of the field.
Pattern	The size of the Pattern is decoded to SIZE, in case no explicit size has been defined so far.

Property	Description
Unique constraint	Unique constraint causes the field to be set as record key.
DefaultValue	DefaultValue is put in Initial.

Additional information:

- The type mappings are only determined by Profiles (no fixed rules)
- Annotation elements are joined and put as BusinessRule
- If the element has a 'SubstitutionGroup', the field is defined as Redefined (the substitute becomes a Redefine)