ISA Dialog Manager

XML Interface

A.06.03.b

This manual depicts the interface for processing XML (Extensible Markup Language) data with the ISA Dialog Manager. It elucidates the objects available for it and their attributes and methods.

ISA Informationssysteme GmbH

Meisenweg 33

70771 Leinfelden-Echterdingen

Germany

Notation Conventions

DM will be used as a synonym for Dialog Manager.

The notion of UNIX in general comprises all supported UNIX derivates, otherwise it will be explicitly stated.

< >	to be substituted by the corresponding value
color	keyword
.bgc	attribute
{ }	optional (0 or once)
[ ]	optional (0 or n-times)
<A> | <B>	either <A> or <B>

Description Mode

All keywords are bold and underlined, e.g.

variable **integer** **function**

Indexing of Attributes

Syntax for indexed attributes:

[I]

[I,J] meaning [row, column]

Identifiers

Identifiers have to begin with an uppercase letter or an underline ('_'). The following characters may be uppercase or lowercase letters, digits, or underlines.

Hyphens ('-') are ***not*** permitted as characters for specifying identifiers.

The maximal length of an identifier is *31* characters.

Description of the permitted identifiers in the Backus-Naur form (BNF)

<identifier>	::=	<first character>{<character>}
<first character>	::=	_ | <uppercase>
<character>	::=	_ | <lowercase> | <uppercase> | <digit>

<digit> ::= 1 | 2 | 3 | … 9 | 0

<lowercase> ::= a | b | c | … x | y | z

<uppercase> ::= A | B | C | … X | Y | Z

Table of Contents

1 XML Support

IDM XML support allows for the processing of XML Documents in the Rule Language. The main components include the new object classes ***document*** and ***doccursor***, which are described in the following sections.

The functionality of XML transformation builds on the XML interface and consists of the objects ***transformer*** and ***mapping***. It serves for transformation of XML files to IDM objects and attributes and vice versa.

XML and the “Document Object Model” (DOM), which defines an object model and an interface for the access of XML Documents, are standards of the World Wide Web Consortium (W3C). The specifications are available on the W3C website (**www.w3c.org/XML**, **www.w3c.org/DOM**).

1.1 XML Support on Microsoft Windows

Microsoft XML Core Services (MSXML) Version 3.0 or higher is required.

1.2 XML Support on Unix

The XML interface is available from IDM version A.05.02.d for Unix platforms that support libxml2 and libxslt.

These are the platforms:

- Solaris (version 7 and above)
- HP-UX 11.0 11.23 (32-bit)
- AIX 5.1
- Redhat6.1, Redhat8
- RHEL4 i386 & x86_64
- Suse 9.1

Please note that version 2.7.2 (HP-UX: 2.7.3) or higher of the libxml2 libraries has to be installed on the system. To use ***Transformer*** objects the libxslt library 1.1.24 has to be available. The libraries don’t have to be explicitly linked but they should be in the library search path (depending on the platform LD_LIBRARY_PATH, SHLIB_PATH, LIBPATH) or directly linked while building an IDM application. The attributes *.idispatch* and *.ixmldomdocument2* return the xmlDocPtr and xmlNodePtr respectively xmlAttrPtr depending on the linked libxml2 library. Setting these attributes is not allowed.

1.3 Differences Between the Windows and Unix Implementations

1. The processing instruction xml is not treated as an distinct node of the DOM by libxml2. This has influence on the path, the traversing with **:select** as well as saving the documents.
2. The handling of spaces may be different. The libxml2 library is used with *keepBlanksDefault = 0* to avoid unnecessary text nodes caused by node indentation or line breaks. Through this a behavior as similar as possible to the Windows implementation is achieved.
3. When reading out the attribute *.text* and saving a formatted document there are small differences between MSXML and libxml2 regarding indentation, line breaks and spaces.

2 Use of XML with the Datamodel

Availability

Since IDM version A.06.01.b

XML may be used as a Data Model, where node texts and node attributes can contain the data – usually strings. An XML Document is linked to a View through a ***doccursor***. For this purpose, Data Model attributes and selection patterns for nodes of the XML Document are defined at the ***doccursor***. Further it can be defined whether the Data Model attribute is linked to the content or an attribute value of the node. For operations that change the XML Document, the data changes are forwarded to all Data Model attributes.

The ***doccursor*** has the following attributes to use it as a Data Model:

Table 1: *Datamodel attributes of the doccursor*

Attribute	Meaning
.dataselect	Defines a Data Model attribute with associated selection pattern.
.dataselectattr	Maps a Data Model attribute to an attribute of the selected nodes.
.dataselecttype	Data type conversion of a Data Model attribute.
.dataselectcount	Defines the cardinality of a Data Model attribute.
.dataoptions	Controls caching via the index *dopt_cache_data*.

The *.dataselect* attribute is of crucial importance here. It defines the Data Model attributes of the ***doccursor*** and their linking to nodes of the XML Document using selection patterns. The syntax of the selection patterns is the same as the pattern definitions for the **:select** method. The selection pattern of a *.dataselect* attribute without index – that is, without a Data Model attribute – is applied before the selection patterns of all Data Model attributes. This enables relative selection patterns for the Data Model attributes originating from the nodes preselected by the *.dataselect* attribute without index. At the same time, this reduces the effort required to collect the data.

When collecting the data for a Data Model attribute, it is looped through the document using the selection pattern. Either the node content is fetched using the *.text* attribute of the ***doccursor*** or the value of the node attribute defined by *.dataselectattr*.

For the Data Model attributes defined with *.dataselect*, data type and cardinality may be altered using the attributes *.dataselecttype* and *.dataselectcount*. By default, the Data Model attributes contain vectors with string values (data type *vector[string]*). With the . *dataselecttype* attribute a data type (e.g. *integer*, *boolean*) can be defined to convert the values into. The cardinality (vector or scalar) of Data

Model attributes can be controlled via the attribute *.dataselectcount*. If the data type of the Data Model attribute is a collection or its cardinality is the data type *integer*, then the Data Model attribute contains a vector, otherwise only a scalar with the first value.

Once retrieved values of a Data Model attribute are cached until either the XML Document or the Data Model attributes of the ***doccursor*** change. This caching can be prevented by setting *.dataoptions [dopt_cache_data] = false* at the ***doccursor***.

Saving data in an XML Document happens in a similar way with reversed operations. However, nodes cannot be created or deleted automatically in the XML Document.

Notes

- Collecting the data can be an “expensive” operation, since the selection pattern must be searched in the whole XML Document. The effort may be reduced by limiting the search to preselected sub-trees by using a *.dataselect* attribute without index.
- A ***doccursor*** used as a Data Model should not be used for other purposes, as it typically changes its selection path constantly.

2.0.1 Example

A list of Nobel laureates shall be read from the following XML file and displayed in a table:

```
<?xml version="1.0"?>
<nobelprizes>
  <category id="p">Physics</category>
  <category id="c">Chemistry</category>
  <winner year="1" category="p">Wilhelm Conrad Röntgen</winner>
  <winner year="11" category="c">Marie Curie</winner>
  <winner year="18" category="p">Max Planck</winner>
  <winner year="70" category="c">Luis Leloir</winner>
</nobelprizes>
```

The names of the laureates are taken from the content of the XML nodes “winner” and stored in the Data Model attribute “.Winner”. The years of the award can be found in the “year” attribute of the XML nodes and are counted in the file from 1900. They are converted to *integer* and read as *vector* into the Data Model attribute “.Year”. In the overwritten **:represent** method, the years are completed to four-digit numbers for display.

```
dialog D

document Doc
{
  doccursor DocCur
  {
    .dataselect[.Winner]    "..winner";
    .dataselect[.Year]      "..winner";
```

```
    .dataselectattr[.Year]  "year";
    .dataselecttype[.Year]  integer;
    .dataselectcount[.Year] integer;
  }
}

window Wi
{
  .title "Nobel prize winners";
  .width 300;  .height 220;

  tablefield Tf
  {
    .xauto 0;  .yauto 0;
    .datamodel DocCur;
    .colcount 2;  .rowcount 1;
    .rowheader 1;  .colheader 1;
    .rowheight[0] 25;
    .colwidth[0] 180;  .colwidth[1] 60;
    .content[1,1] "Year";
    .content[1,2] "Winner";
    .dataget[.field]     .Winner;
    .dataget[.userdata]  .Year;
    .dataindex[.userdata] [0,1];

    :represent()
    {
      variable integer I;
      if Attribute=.userdata then
        for I := 1 to itemcount(Value) do
          Value[I] := 1900 + (Value[I] % 100);
        endfor
        setvector(this, .content, Value,[2,1],
                  [1 + itemcount(Value),1]);
        return;
      endif
      pass this:super();
    }
  }

  on close { exit(); }
}

on start
{
  Doc:load("nobelprizes.xml");
}
```

This dialog produces the following window:

Figure 1: *Table with XML data used as Data Model*

2.0.2 Index Value *dopt_cache_data* of the Attribute *dataoptions*

Attribute Index	Default	Component	Meaning
dopt_cache_data	***true***	Model	This index value is only available for the ***doccursor***. **true** The data selected by the ***doccursor*** is buffered for further accesses (“caching”). **false** With each access, the ***doccursor*** selects the data anew from the XML Document.

3 The XML Document (document)

The document object is the container for an XML Document. An XML Document is saved as a DOM tree. This DOM tree can be traversed with the help of a doccursor, which must be a child of the document object.

Definition

```
{ export | reexport } { model } document { <Identifier> }
{
  [ <atribute definition> ]
  [ <method definition> ]
}
```

Events

None

Children

doccursor

document

record

transformer

Parents

application

canvas

checkbox

dialog

doccursor

document

edittext

groupbox

image

import

layoutbox

listbox

menubox

menuitem

menusep

messagebox

module

notebook

notepage

poptext

pushbutton

radiobutton

record

rectangle

scrollbar

spinbox

splitbox

statictext

statusbar

tablefield

timer

toolbar

transformer

treeview

window

Menu

None

Methods

:load()

:save()

:transform()

:validate()

3.1 Attributes

doccursor[l]

document[I]

external

external[I]

firstrecord

idispatch

ixmldomdocument2

label

lastrecord

model

parent

real_version[enum]

record[I]

recordcount

scope

transformer[I]

userdata

version[enum]

xml

3.2 Object-specific Attributes

doccursor[I]

It is possible to access the XML Cursor of the XML Document through the doccursor attribute. The attribute is indexed with the object index (similar to child).

idispatch

The Idispatch COM interface pointer in XML Documents can be accessed through the idispatch attribute under Microsoft Windows.

In the Rule Language the attribute can only be assigned to the same attribute of a different IDM object. Please note, in the programming interface a COM object will remain valid only as long as it remains in use in the ISA Dialog Manager. For this reason it is important to increase the reference counter of an application (COM Method: IUnknown->AddRef) . When the object is no longer needed, the counter must be decreased again (COM Method: IUnknown->Release). In any case, it is not allowed to decrease the counter more than it is increased. If this happens, the COM object will be released. The ISA Dialog Manager cannot recognize this situation, which will lead to a sys-

tem crash. The ISA Dialog Manager may also crash when the given pointer does not point to a COM interface.

This attribute is not passed down because it refers to a runtime characteristic.

ixmldomdocument2

The IXMLDOMDocument2 COM interface pointer in XML Documents can be accessed through the ixmldomdocument2 attribute under Microsoft Windows.

In the Rule Language the attribute can only be assigned to the same attribute of a different IDM object. Please note, in the programming interface a COM object will remain valid only as long as it remains in use in the ISA Dialog Manager. For this reason it is important to increase the reference counter of an application (COM method: IUnknown->AddRef). When the object is no longer needed, then the counter must be decreased again (COM Method: IUnknown->Release). In any case, it is not allowed to decrease the counter more than it is increased. If this happens, the COM object will be released. The ISA Dialog Manager cannot recognize this situation, which will lead to a system crash. The ISA Dialog Manager may also crash when the given pointer does not point to a COM interface.

This attribute is not passed down because it refers to a runtime characteristic.

xml

The string representation of XML Documents can be accessed with this attribute. When a new value is set, the saved DOM tree is deleted and a new DOM tree is built from the newly set value. All existing XML Cursors become invalid. When an XML Cursor is invalid, the attribute *.mapped* has the value *false*.

3.3 Object-specific Methods

:load()

This loads an XML Document from the given file or URL. The saved DOM tree is deleted, and a new DOM tree is built. All existing XML Cursors become invalid. When an XML Cursor is invalid, the attribute *.mapped* has the value *false*.

:save()

Saves the XML Document to a file or URL.

:transform()

Transforms the XML Document with the given scheme. When the target is an XML Document, the saved DOM tree is deleted and a new tree is built. All existing XML Cursors become invalid. When an XML Cursor is invalid, the attribute *.mapped* has the value *false*.

Alternatively, the target of the transformation can be a text or file. If the result of the transformation is no legal XML format, then it needs to be directly converted into a text or file, as the result cannot be assigned to an XML Document. This is true, for example, for conversions to HTML.

:validate()

This checks if the XML Document conforms to the document type given in the XML Document. If it does not have a document type, an error will occur.

4 The XML Cursor (doccursor)

The ***doccursor*** object is always a child of an XML Document. It refers to a node in the DOM tree, which is saved in the XML Document (the parent).

There are methods to set the reference to a different node of the DOM tree. In other words this means that the XML Cursor can be moved in the DOM tree through methods. It will remain a child of the XML Document of course.

Definition

```
{ export | reexport } { model } doccursor { <Identifier> }
{
  [ <atribute definition> ]
  [ <method definition> ]
}
```

Besides the “normal” ISA Dialog Manager attributes, the XML Cursor attribute contains additional attributes which make it possible to access properties such as name, value or other attributes of the DOM nodes. Because these are runtime attributes, they cannot be passed down. In addition, many of these attributes can only be read because the corresponding properties of the DOM nodes cannot be changed.

The XML Cursor is initially invalid, but as soon as it is accessed for the first time it will be automatically positioned to the root of the DOM tree. Note that this also happens when the XML Cursor gets invalid through an action. The attribute *.mapped* shows if the XML Cursor is valid or not.

Events

None

Children

document

record

transformer

Parents

document

Menu

None

Methods

:add()

:delete()

:match()

:reparent()

:select()

:transform()

4.1 Attributes

attribute[I]

attribute[string]

data

dataselect[attribute]

dataselectattr[attribute]

dataselectcount[attribute]

dataselecttype[attribute]

document[I]

external

external[I]

firstrecord

idispatch

ixmldomnode

ixmldomnodelist

label

lastrecord

mapped

model

name

nodetype

parent

path

publicid

record[I]

recordcount

scope

specified

systemid

target

text

transformer[l]

userdata

value

xml

4.2 Object-specific Attributes

attribute[l]
attribute[string]

Depending on the indexing, the attribute “attribute” serves for querying the name or the value of a DOM nodes’ attributes .

If the index is a number, the name of the corresponding attribute of a DOM node will be delivered. Take note that attributes of DOM nodes usually are unsorted.

If the index is a string, then the index is regarded as a name of an attribute and the value of this attribute is returned. An assignment to an attribute indexed with a string creates a corresponding attribute for the DOM node. The assignment of an empty string deletes the corresponding attribute of the DOM node.

This attribute is not passed down because it refers to a runtime characteristic.

data

Is for setting and retrieving the data within a DOM node. The attribute is only available when the node type is either *nodetype_cdata_section* or *nodetype_processing_instruction*.

This attribute is not passed down because it refers to a runtime characteristic.

idispatch

The IDispatch COM interface pointer of the XML Cursor can be accessed through the attribute idispatch under Microsoft Windows.

In the Rule Language the attribute can only be assigned to the same attribute of a different IDM object. Be aware, that in the programming interface the COM object is only valid during the time when it is in use by the ISA Dialog Manager. Therefore, an application should increase its reference counter (COM Method: IUnknown->AddRef). When the object is no longer needed, the counter should be decreased (COM Method: IUnknown->Release). Please note, the counter cannot be decreased more often than it is increased. Otherwise, the COM object will be released. The ISA Dialog Manager cannot recognize this situation and this will lead to a system crash.

This attribute is not passed down because it refers to a runtime characteristic.

ixmldomnode

The IXMLDOMNode COM interface pointer of the XML Cursor can be accessed through the ixmldomnode attribute under Microsoft Windows.

In the Rule Language the attribute can only be assigned to the same attribute of a different IDM object. Be aware, that in the programming interface the COM object is only valid while it is in use by the ISA Dialog Manager. Therefore, an application should increase its reference counter (COM Method: IUnknown->AddRef). When the object is no longer needed, the counter should be set back again, or decreased (COM Method: IUnknown->Release). Please note, the counter cannot be decreased more often than it is increased. Otherwise, the COM object will be released. The ISA Dialog Manager cannot recognize this situation and this will lead to a system crash.

This attribute is not passed down because it refers to a runtime characteristic.

ixmldomnodelist

The IXMLDOMNodeList COM interface pointer of the XML Cursor can be accessed through the ixmldomnodelist attribute under Microsoft Windows. The direct children of the XML Cursor can be accessed through the interface pointer.

In the Rule Language the attribute can only be assigned to the same attribute of a different IDM object. Be aware, that in the programming interface the COM object is only valid while it is in use by the ISA Dialog Manager. Therefore, an application should increase its reference counter (COM Method: IUnknown->AddRef).). When the object is no longer needed, the counter should be decreased (COM Method: IUnknown->Release). Please note, the counter cannot be decreased more often than it is increased. Otherwise, the COM object will be released. The ISA Dialog Manager cannot recognize this situation and this will lead to a system crash.

This attribute is not passed down because it refers to a runtime characteristic.

mapped

Is true when the XML Cursor points to a node in the DOM tree. Please take note that an XML Cursor, which does not reference any node in the DOM tree, is automatically positioned on the root of the DOM tree when any object-specific attribute is accessed or when an object-specific method is invoked.

This attribute is not passed down because it refers to a runtime characteristic.

name

Refers to the name or tag of the DOM node.

This attribute is not passed down because it refers to a runtime characteristic.

nodetype

This serves for querying the type of DOM node.

This attribute is not passed down because it refers to a runtime characteristic.

path

Delivers a string representation for the position of the XML Cursor in the DOM tree. The select method can be called with this string in order to position an XML Cursor to the nodes in the DOM tree.

If the value of the path attribute is stored somewhere else (i.e. in the userdata attribute), then it is important to know that these stored values will not be adjusted when the structure of the DOM tree changes. When the select method is invoked with the stored value afterward, the XML Cursor will point to an incorrect DOM node.

This attribute is not passed down because it refers to a runtime characteristic.

publicid

Is the public identifier of the DOM node. The attribute is only available, when the node type is either *nodetype_entity* or *nodetype_notation*.

This attribute is not passed down because it refers to a runtime characteristic.

specified

This reveals if an attribute of a DOM node was explicitly given, or if it was inherited from a standard value. The attribute is always *true* except for the node type *nodetype_attribute*.

This attribute is not passed down because it refers to a runtime characteristic.

systemid

Is the system identifier of DOM nodes. This attribute is only available when the node type is either *nodetype_entity* or *nodetype_notation*.

This attribute is not passed down because it refers to a runtime characteristic.

target

Is the name of the instruction for a DOM node. The value is the same as the value of the name attribute. This attribute is only available when the node type is *nodetype_processing_instruction*.

This attribute is not passed down because it refers to a runtime characteristic.

text

Is the value of all sub-nodes within a DOM node. A string is delivered which represents the text of all sub-nodes. This attribute is mainly helpful when only the text of an XML element is needed, as it is not required to navigate to the child nodes containing the actual text.

Please note that setting this attribute automatically deletes all child nodes and inserts a new text node.

This attribute is not passed down because it refers to a runtime characteristic.

value

Is the value of a DOM node. This attribute is only available when the node type is *nodetype_attribute*, *nodetype_text*, *nodetype_cdata_section*, *nodetype_processing_instruction* or *nodetype_*

comment.

This attribute is not passed down because it refers to a runtime characteristic.

xml

String representation of a DOM node and all of its child nodes.

This attribute is not passed down because it refers to a runtime characteristic.

4.3 Object-specific Methods

:add()

Inserts a child node as the last node to the current DOM node. The XML Cursor is then set to the new element.

:delete()

Deletes the DOM node and all of its child nodes. The XML Cursor is then positioned to the father node. The XML Cursors, which point to the deleted DOM nodes in the sub-tree, become invalid. The attribute *.mapped* has the value *false* for invalid XML Cursors.

When the value of the path attributes is stored elsewhere (i.e. in userdata attribute), then it is important to know that these stored values will not be adjusted when the structure of the DOM tree changes. When the select method is invoked with the stored value afterward, the XML Cursor will point to an incorrect DOM node.

:match()

Tests if the DOM node satisfies the given pattern (see chapter “Pattern for the Methods :match() and :select()”).

:reparent()

Reallocates the DOM node with all of its child nodes.

When the value of the path attributes is stored elsewhere (i.e. in userdata attribute), then it is important to know that these stored values will not be adjusted when the structure of the DOM tree changes. When the select method is invoked with the stored value afterward, the XML Cursor will point to an incorrect DOM node.

:select()

Moves the XML Cursor in the given direction or moves the XML Cursor to the first DOM node that matches the given pattern (see chapter “Pattern for the Methods :match() and :select()”).

:transform()

Transforms the XML Cursor with the given scheme. When the target is an XML Document, the saved DOM tree is deleted, and a new tree is built. All existing XML Cursors become invalid. The attribute *.mapped* has the value *false* for invalid XML Cursors.

Alternatively, the target can be a text or file. If the result of the transformation is no legal XML format, then it needs to be directly converted into a text or file, as the result cannot be assigned to an XML Document. This is true, for example, for conversions to HTML.

4.4 Pattern for the Methods :match() and :select()

A pattern is very similar to a ISA Dialog Manager identifier. The pattern represents a path of element names. The path starts at the root of the tree. Each hierarchical level is compared to the corresponding part of the path. Here it is possible to define certain characteristics of the father such as existence of an attribute or the position within the children.

Basically all characters except **.**, **[**, **"**, **]**, **<**, **>**, **=**, \, tab, space and line break can be used in the pattern. If one of the characters mentioned above, without the page break, shall be used, then it has to be escaped with a \ before the character. Between double quotes (**"**), that is within a string, the following symbols are also allowed: **.**, **[**, **]**, **<**, **>**, **=**, tab and line break. Please note that in the dialog script the symbol \ within a string already is an escape symbol. Due to this it is important to use **** in a dialog script whenever a \ is needed. Also in a dialog script **\"** has to be used whenever the character **"** is needed.

```
{ <Name> [[.<Attr>{<Op>"<Value>"}]] {[<Idx>]} }
  [ .<Name> [[.<Attr>{<Op>"<Value>"}]] {[<Idx>]} ]
```

<Name>

Is compared to the name attribute of the XML Cursor. The XML Cursor must possess the node type *nodetype_element*. Alternatively, the character * can be used. In this case the name attribute will not be taken into consideration.

The name of an XML element starts with a letter or an underline. It may contain letters, digits, hyphens, underlines and dots. The exact definition of an XML element name can be found in the XML specifications (**www.w3c.org/XML**).

<Attr>

The DOM node that points to the XML Cursor must possess the given attribute.

The name of an XML attribute starts with a letter or an underline. It may contain letters, digits, hyphens, underlines and dots. The exact definition of an XML attribute name can be found in the XML specifications (**www.w3c.org/XML**).

<Op>

Is a comparison operator which compares the attribute given in <Attr> with <Value>. It can be tested for equality **=** and inequality **<>**.

<Value>

Is the value to which the <Attr> is compared.

<Idx>

The DOM node must be at this position within the child nodes of the same parent. The first child has position 1.

<Idx> consists only of figures (**0** – **9**), which are interpreted as number.

Particularities

.

When the pattern start with a dot, it is relative to the current DOM node. This means that the path begins at the current DOM node.

..

Two consecutive dots define, that an arbitrary number of hierarchy levels may be skipped.

[<Idx>]

An index without any further information selects the DOM node at this position. In this case, the type of DOM node is insignificant. Thus each DOM node can be uniquely referenced by an expression like {**[**<Idx>**]**[**.[**<Idx>**]**]} (path attribute).

Additions for Microsoft Windows

XPath can also be used as pattern syntax. Here, the pattern must start with either **/** or **./**. The use of XPath patterns is not supported on all platforms and therefore not portable.

5 The transformer Object

The transformer object allows for traversing an XML Document or an IDM hierarchy, whereby during the traversing semantic actions can be carried out on particular nodes. Through this it is easy to implement a transformation of data. For example, XML data can be transferred into IDM objects or vice versa, XML Documents can be generated from the data contained in IDM objects. Because semantic actions are described through user-defined code, the transformations can have many different forms.

Definition

```
{ export | reexport } { model } transformer { <Identifier> }
{
  [ <atribute definition> ]
  [ <method definition> ]
}
```

The following means are available to the IDM programmer regarding the definition of transformations.

- A transformation is started by invocation of the apply method of a transformer object. The source and the target of the transformation are passed as parameters, (i.e. a doccursor object as source and an IDM object as target, which either collects the data or delegates it elsewhere).
- It is normally desired that during each transformation a certain amount of nodes, regardless if these are IDM objects or nodes of an XML tree, are run through in a certain sequence. The apply method implements such a sequence by beginning with a start node (source) and calling the transformer's select_next method, which determines the successor of the actual node, in a loop. The sequence ends when the select_next method returns 0. The default implementation of the select_next method defines a pre-order sequence. The IDM programmer is able to intervene at two different points in this process. First of all, the apply method can be overwritten in order to set the start node or to change the abort condition. Secondly, the select_next method can be overwritten and therefore the entire sequence in which the nodes are visited can be redefined.
- After it has been guaranteed that all nodes are visited at least once, then a possibility must exist for setting the semantic action(s) to be carried out on certain nodes. For this purpose mapping objects are defined as children of the transformer object. Each of these objects defines a pattern in its name attribute, which is used to check if the currently visited node should trigger an action or not. The action is set through the action method of the mapping object that has to be overwritten by the IDM programmer. This method receives the currently visited node as first parameter, and the target object coming from the apply method of the transformer object as second parameter.
- In order to complete the picture: The transformer object also has an action method. This method is called for each visited node within the loop of the apply method mentioned above. It is then checked if the node matches the patterns of the mapping children. The sequence in which this happens is the same sequence as the mappings have been defined in the parent transformer. The inherited mapping objects are examined at the very end. If a matching mapping object is found

during the examination, the action method of the transformer calls the action method of the mapping object. This method can transfer the data from the current node to the target object. As a return value, this method can return *true*. In this case it is assumed that the node has been processed completely and no further mapping objects should be examined. Otherwise, the remaining mappings are examined and their action methods are called until no more mapping objects are left or the action method of one of the mappings returns *true*.

Events

None

Children

document

mapping

record

transformer

Parents

application

canvas

checkbox

dialog

doccursor

document

edittext

groupbox

image

import

layoutbox

listbox

mapping

menubox

menuitem

menusep

messagebox

module

notebook

notepage

poptext

pushbutton

radiobutton

record

rectangle

scrollbar

spinbox

splitbox

statictext

statusbar

tablefield

timer

toolbar

transformer

treeview

window

Menu

None

Methods

:action()

:apply()

:select_next()

5.1 Attributes

document[I]

external

external[I]

firstrecord

label

lastrecord

mapping[I]

model

module

parent

record[I]

recordcount

root

scope

userdata

5.2 Object-specific Attributes

mapping[I]

The mapping children can be accessed through the .mapping attribute. The attribute is indexed with the object index (similar to child). The sequence of the mappings within this vector determines the sequence in which nodes are compared to particular mappings during a transformation. The inherited mappings are not taken into this vector. During a transformation, the comparison to these mappings is done at the very end, i.e. the direct instances have priority. This is different in comparison to other inherited child objects within IDM, which are inserted at the beginning of the parent instance's child vector.

root

After calling the **:apply()** method the starting point of the transformation will be stored in this attribute. This makes it possible to decide during a transformation, if the starting point has been reached again, and if the transformation can be stopped.

The following cases should be distinguished:

- When the Src parameter of the apply method is a document or a doccursor, a string is saved in .root, which describes the corresponding position in the XML tree. The syntax of the string follows the convention, which is used for the .path attribute of the doccursor object (see doccursor object description). Because of this, comparisons to the .path attributes of doccursors are very easy.
- If the Src parameter of the apply method is an IDM object, this object will be stored in .root. Consequently, in this case, the data type of the attribute is object.

On the basis of the value in the .root attribute, the action and select_next methods of the transformer decide what they have to do (see description of these methods).

At the end of the apply method *.root* is set to *void* again.

The default value for the attribute is *void*.

5.3 Object-specific Methods

:apply()

The transformation is initiated with this method. The default implementation of the algorithm is as follows:

» If the Src parameter is a document or a doccursor object, it is assumed that data from the XML tree should be transferred to the IDM. In the case of a document object, a temporary doccursor object will be created which is used for the navigation within the XML tree. In the pseudo-code below, for the purpose of simplicity it is assumed that a doccursor is passed in Src.

```
:apply(anyvalue Src, anyvalue Dest)
{
  variable object NextObj;

  this.root ::= Src.path;
  NextObj := Src;
  while NextObj <> null do
    this:action(Src, Dest);
    NextObj := this:select_next(NextObj);
  endwhile
  this.root ::= null;
  return true;
}
```

» If the Src parameter is an IDM object (except for document and doccursor objects), it is assumed that data from IDM should be transferred to XML or somewhere else. The underlying code is identical to the code above, except that Src is stored in .root instead of Src.path. For example:

```
this.root ::= Src;
```

The apply method can be overwritten (similar to init).

:action()

This method is used by the apply method of the transformer (see above) to determine if the current node matches one of the mapping children. If it does, the action method of the mapping object is called.

The action method can be overwritten (similar to init).

:select_next()

This method is used by the apply method of the transformer for traversing all nodes of an XML tree or an IDM object hierarchy (see above). The default implementation of this is, that a pre-order sequence is processed by the repeated calls of the method.

The select_next method can be can be overwritten (similar to init).

5.4 Example

This example can be found in the *examples/xml* directory.

As XML Document the following file with the name “CD-Katalog.xml” is used:

```
<?xml version="1.0" ?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
  <CD>
    <TITLE>Maggie May</TITLE>
    <ARTIST>Rod Stewart</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Pickwick</COMPANY>
    <PRICE>8.50</PRICE>
    <YEAR>1990</YEAR>
  </CD>
</CATALOG>
```

The data from this file can, for example, be read out as follows:

```
dialog D {}

window Wi
{
  .title "XML-CD-CATALOG";

  on close { exit(); }

  child treeview Tv
  {
    .xauto 0;
    .yauto 0;
    .style[style_lines]   true;
```

```
    .style[style_buttons] true;
    .style[style_root]    true;
    integer CdIdx := 0;

    rule NewCatalog() {
      if this.itemcount = 0 then
        this.itemcount ::= this.itemcount + 1;
      endif
      this.content[this.itemcount] := "CD-Catalog";
      this.open[this.itemcount] := true;
    }

    rule AddCD() {
      this:insert(this.itemcount+1, 4);
      this.CdIdx := this.CdIdx + 1;
      this.content[this.itemcount-3] := ""+this.CdIdx+". CD";
      this.level[this.itemcount-3] := 2;
    }

    rule AddTitle(string S input) {
      this.content[this.itemcount-2] := "Title: " + S;
      this.level[this.itemcount-2] := 3;
    }

    rule AddArtist(string S input) {
      this.content[this.itemcount-1] := "Artist: " + S;
      this.level[this.itemcount-1] := 3;
    }

    rule AddPrice(string S input) {
      this.content[this.itemcount] := "Price: " + S;
      this.level[this.itemcount] := 3;
    }
  }
}

transformer Tr
{
  !! transformer is for taking over data from an XML Document
  !! therefore a doccursor can be expected in Src

  child mapping MCatalog {
    .name "..CATALOG";

    :action() {
      Dest:NewCatalog();
      return true;
```

```
    }
  }

  child mapping MCD {
    .name "..CD";

    :action() {
      Dest:AddCD();
      return true;
    }
  }

  child mapping MTitle {
    .name "..CD.TITLE";

    :action() {
      Dest:AddTitle(Src.text);
      return true;
    }
  }

  child mapping MArtist {
    .name "..CD.ARTIST";

    :action() {
      Dest:AddArtist(Src.text);
      return true;
    }
  }

  child mapping MPrice {
    .name "..CD.PRICE";

    :action() {
      Dest:AddPrice(Src.text);
      return true;
    }
  }
}

document Doc {}

on dialog start
{
  Doc:load("CD-Katalog.xml");

  Tv.visible := false;
```

```
  Tr:apply(Doc, Tv);
  Tv.visible := true;
}
```

6 The mapping Object

The purpose of the mapping object is to define a semantic action, which is called during a transformation for a specific node in an XML tree or in an IDM object hierarchy), when a match between the mapping object and the node is found. Mapping objects are defined as children of a transformer object and describe a transformation of data in conjunction with it.

Definition

```
{ export | reexport } { model } mapping { <Identifier> }
{
  [ <atribute definition> ]
  [ <method definition> ]
}
```

Events

None

Children

document

record

transformer

Parents

transformer

Menu

None

Methods

:action()

6.1 Attributes

document[I]

external

external[I]

firstrecord

label

lastrecord

name

model

parent

record[I]

recordcount

scope

transformer[I]

userdata

6.2 Object-specific Attributes

name

Here, a pattern is given (similar to an XPath expression) to define the nodes within an XML tree or an IDM object hierarchy which the mapping object should match. This determines if the :action method should be called for the node during the transformation or not.

6.3 Pattern for .name

A pattern is very similar to an IDM identifier. The pattern represents a path of element names. The path starts at the root of the document or IDM object hierarchy (i.e. dialog or module). Each hierarchy level is compared to the corresponding part of the path. In addition, certain properties of the parent, such as the existence of an attribute or the position within the children or the parent can be given:

```
{ <Name> [[.<Attr>{<Op>"<Value>"}]] {[<Idx>]} }
  [ .<Name> [[.<Attr>{<Op>"<Value>"}]] {[<Idx>]} ]
```

<Name>

XML: Is compared to the name attribute of the cursor. The cursor must have the node type *node-type_element*. Alternatively, * can be used. In this case the name attribute is ignored.

IDM: Is compared to the label of an IDM object. Alternatively, * can be used, which means that every label matches.

<Attr>

XML: The XML node that the cursor points to must have the indicated attribute.

IDM: The IDM object must have the indicated attribute.

<Op>

XML and **IDM:** Comparison operator which compares the attribute given in <Attr> with <Value>. I can be tested for equality **=** and inequality **<>**.

<Value>

XML and **IDM:** The value to which <Attr> is compared.

<Idx>

XML: The XML node must be at this position within the child nodes of the same parent.

IDM: The IDM object must be at this position within the children of the same parent. Here, all children of hierarchical attributes are regarded as combined.

Particularities

.

XML: If the pattern starts with a dot, it is relative to the current node. This means that the path begins at the current node.

..

XML and **IDM:** Two consecutive dots define, that an arbitrary number of hierarchy levels may be skipped.

[<Idx>]

XML: An index, without any further information, selects the XML node at this position. In this case, the type of node is insignificant. Thus each DOM node can be uniquely referenced by an expression like {**[**<Idx>**]**[**.[**<Idx>**]**]} (path attribute).

All comparisons are case sensitive.

Example

The pattern "..CD[.Title = Yellow]" matches all nodes within an XML tree (regardless of their hierarchy level) that have the tag "CD" and an attribute ".Title" with the value *Yellow*.

When the same pattern is applied to IDM objects, all objects possessing the label "CD", and an user-defined attribute ".Title" with the value *Yellow* will match.

6.4 Object-specific Methods

:action()

This method is invoked from the **:action** method of the parent ***transformer***, when a match between a node in an XML tree or an IDM object hierarchy and the pattern in the *.name* attribute of the ***mapping*** objects is found. The default implementation of this method does nothing other than always returning *true*.

Because this method can be redefined, it is possible for the programmer to determine what should happen with the data from the node.

Index

N

O

P

R

S

T

V

X