Introduction

The following topics are covered:

- XML Toolkit Features
- XML Toolkit Description
- Considerations and Limitations

XML Toolkit Features

- Natural-based XML parser using dynamic variables.
- Functions for
  - conversion of Natural data structures into DTD definitions;
  - generation of COMPRESS statements to save a Natural data structure as an XML document;
  - generation of callback for the Natural-based parser.

XML Toolkit Description

Objective

The objective of the Natural XML Toolkit is to provide additional XML functionality with Natural and improve the integration of Natural applications with XML.

General Architecture

The Natural XML Toolkit is implemented as a Natural plug-in. The Toolkit programs may be integrated into customer applications to provide access to XML data or to deliver data from Natural in XML format.

The Natural XML Toolkit calls the functions listed below:

XML Toolkit Functions

1. Mapping of Natural Data Definition to DTD or XML Schema and vice versa.
2. **XML Token => NAT**
   Data After the Natural data structure has been created, the XML document has to be parsed and saved into the data structure. A Natural implementation is generated that is capable of saving the given data into the Natural data structure.
3. **NAT Data => XML Document ("Serialize")**
   Serialization is the process of taking the data stored in the Natural data structures and creating an XML document according to the description provided.

A Natural dialog implements the user interface to the XML Toolkit functions. The DTD or XML Schema will be accessed as a work file and the generated Natural objects will be saved directly to the Natural system file.

Map Natural Data Definitions to DTD

This mapping is the first step to bind Natural data structures to XML tags and is required to implement a representation of Natural data as XML tags. The example below shows the mapping as well as some obvious differences between Natural and a DTD.

Natural PDA

```
                       Press ESC to enter command mode
 Mem: EMPL     Lib: SYSXTK  Type: PARAMETER  Bytes:  1072  Line:    0 of:  26
C T   Comment
  *   *** Top of Data Area ***
    1 EMPLOYEE
    2 ATTRIBUTES_OF_EMPLOYEE
    3 PERSONNEL-ID                    A          8
  *
    2 FULL-NAME
    3 FIRST-NAME                      A         20
    3 NAME                            A         20
  *
    2 FULL-ADDRESS
    3 C@ADDRESS-LINE                  I          4
    3 ADDRESS-LINE                    A         20 (1:6)
    3 CITY                            A         20
    3 ZIP                             A         20
    3 COUNTRY                         A          3
  *
    2 TELEPHONE
    3 AREA-CODE                       A          6
    3 PHONE                           A         15
```

Generated DTD

```
<!ELEMENT EMPLOYEE (PERSONNEL-ID, FULL-NAME, FULL-ADDRESS, TELEPHONE, INCOME* )>

<!ELEMENT PERSONNEL-ID (#PCDATA ) >

<!ELEMENT FULL-NAME (FIRST-NAME, NAME )>
         <!ELEMENT FIRST-NAME  (#PCDATA )>
         <!ELEMENT NAME  (#PCDATA )>

<!ELEMENT FULL-ADDRESS (ADDRESS-LINE*, CITY, ZIP, COUNTRY )>
         <!ELEMENT ADDRESS-LINE  (#PCDATA )>
         <!ELEMENT CITY  (#PCDATA )>
         <!ELEMENT ZIP  (#PCDATA )>
         <!ELEMENT COUNTRY  (#PCDATA )>
...
```

The generated DTD will be used later on during serialization to a XML document (see below).

Serialize Data to XML

During execution of a Natural program, the content of the data defined in the DEFINE DATA statement will be filled with "real" content. This content will be written to a dynamic variable in XML format during serialization and will use the formerly generated DTD as input.

The XML Toolkit generates the program to serialize the data.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<EMPLOYEE PERSONNEL-ID="30016509">
<FULL-NAME>
  <FIRST-NAME>ELSPETH</FIRST-NAME>
  <NAME>TROWBRIDGE</NAME>
</FULL-NAME>
<FULL-ADDRESS>
  <ADDRESS-LINE>91 BACK LANE</ADDRESS-LINE>
  <ADDRESS-LINE>BILSTON</ADDRESS-LINE>
  <ADDRESS-LINE>STAFFORDSHIRE</ADDRESS-LINE>
  <CITY>BILSTON</CITY>
  <ZIP>ST2 3KA</ZIP>
  <COUNTRY>UK</COUNTRY>
</FULL-ADDRESS>
<TELEPHONE>
  <PHONE>863322</PHONE>
  <AREA-CODE>0602</AREA-CODE>
</TELEPHONE>
...
```

Map DTD to Natural Data Definitions

The mapping of a DTD to Natural data structures again shows differences. The DTD does not specify how many person records will be included in the XML document, therefore the Toolkit assumes that a maximum number of "v" persons will be included. The application programmer might know the exact number and the data structure could be adapted accordingly. A similar limitation exists with the length of the data. The DTD does not include information about the length of the data in a person’s record. Therefore the Toolkit creates fields in the data structure with a length of A dynamic, the current maximum.

```
* DTD E:\SAG\nat\6.3\fnat\SYSXTK\RES\empl.dtd
COMPRESS &1& ’<EMPLOYEE’
  ’ PERSONNEL-ID="’EMPLOYEE.PERSONNEL-ID "’
  ’>’ INTO &1& LEAVING NO
/* now the children
COMPRESS &1& ’<FULL-NAME’
  ’>’ INTO &1& LEAVING NO
/* now the children
COMPRESS &1& ’<FIRST-NAME’
  ’>’
  EMPLOYEE.FIRST-NAME
  ’</FIRST-NAME>’ INTO &1& LEAVING NO
COMPRESS &1& ’<NAME’
  ’>’
  EMPLOYEE.NAME
  ’</NAME>’ INTO &1& LEAVING NO
/*
COMPRESS &1& ’</FULL-NAME>’ INTO &1& LEAVING NO
COMPRESS &1& ’<FULL-ADDRESS’
  ’>’ INTO &1& LEAVING NO
/* now the children
```

```
FOR &2& = 1 TO EMPLOYEE.C@ADDRESS-LINE
  COMPRESS &1& '<ADDRESS-LINE'
    '>'
    EMPLOYEE.ADDRESS-LINE(&2&)
    '</ADDRESS-LINE>' INTO &1& LEAVING NO
END-FOR
...
```

Parse XML File and Assign to Natural Data

```
* DTD E:\SAG\nat\6.3\fnat\SYSXTK\RES\empl.dtd
DECIDE ON FIRST &1&
  VALUE 'EMPLOYEE'
    RESET INITIAL EMPLOYEE
  VALUE 'EMPLOYEE/@PERSONNEL-ID'
    /* #REQUIRED
    EMPLOYEE.PERSONNEL-ID := &3&
  VALUE 'EMPLOYEE/FULL-NAME'
    IGNORE
  VALUE 'EMPLOYEE/FULL-NAME/FIRST-NAME'
    IGNORE
  VALUE 'EMPLOYEE/FULL-NAME/FIRST-NAME/$'
    EMPLOYEE.FIRST-NAME := &3&
  VALUE 'EMPLOYEE/FULL-NAME/NAME'
    IGNORE
  VALUE 'EMPLOYEE/FULL-NAME/NAME/$'
    EMPLOYEE.NAME := &3&

...
```

Considerations and Limitations

The XML Toolkit only supports fully assembled XML Schema (Layer 1). For detailed information, refer to the W3C recommendation on XML Schema (Layer 1)

When using the XML Toolkit, the following further limitations should be considered.

- Very Large Data Structures
- Multi-Dimensional Arrays
- XML Schema: Access and Composition
- DTD: Add external parsed data
- Conditional DTD
- Wildcards

Very Large Data Structures

Data structures which will result in more then approximately 700 data fields and groups will end up with the message:

```
Input Structure too big
```

Solution

Split up the data structure into smaller sections.

Multi-Dimensional Arrays

The following limitations apply, when generating an XML document from a Natural data area, if the source data contains arrays:

- For each array, exactly one dimension is allowed.
- For each level, exactly one dimension can be added.
- Each array must have a counter variable.
- The counter variable
  - must be located before the array and
  - the counter variable name must start with the character C followed by the counter separator field.
- The *LBOUND (lower boundary) of the array must be 1.

XML Schema: Access and Composition

<include>

Include adds multiple schemas with the same target namespace to a document. The document needs to be included, without any changes.

<import>

Import adds multiple schemas with different target namespaces to a document. First the document to be imported requires a namespace prefix translation, then the document can be included.

<redefine>

Redefine selects out specific simple and complex types, groups, and attribute groups from an external schema, and enables you to modify the given specification for your own needs.

Note:
With all of the above elements, only relative URIs are allowed. Absolute URIs (e.g. *http://www.yourdomain.com/your/path* or *file://your/path*) can not be used.

DTD: Add external parsed data

The external data has to be included into the document. There are no conversions necessary.

Conditional DTD

If a `<![ INCLUDE ]` is found, the contained definition will be used for generation.

If a `<![ IGNORE ]` is found, the contained definition will not be used for generation.

Wildcards

The XML Toolkit supports two different kinds of wildcard representations:

- Save all subsequent elements or
- Save all attributes that are not specified

For wildcard support the following rules and/or limitations apply.

XML Schema <anyAttribute>

For attributes an `attributes_of_<entity-name>` group is generated. All attributes connected to this group are added. The name of an attribute is saved as a variable name; the content is the content of the variable.

To add `<any>` attributes, it is required to add a variable that contains all attributes not specified.

Note:
The `<any>` attributes are no "real" attributes; they are used as a container for the not parsed data and contain the attribute/value pairs. An `<any>` attribute is represented by a `##ANY` variable of type (A) dynamic.

Because it can be necessary to access this data, a more specific name should be used instead of `##ANY` followed by a generic number. It is recommended to add the name of the parent entity and the keyword `ATTR`, or `ATTRIBUTE`. See below for an example:

```
1 HTML
  2 BODY
    3 ATTRIBUTES_OF_BODY
      4 BGCOLOR         (A) DYNAMIC
      4 ##ANY_ATTR_BODY (A) DYNAMIC
```

If, during the parse process, an attribute that is not named inside the XML Schema is found, the variable name and value will be saved at the `all_attributes_of_<element-name>` group as they are, this means with the standard XML syntax:

`<attribute-name1>="<attribute-value1>"`
`<attribute-name2>="<attribute-value2>"` etc.

When serializing, the above string will be added.

XML Schema <any> or DTD <!ELEMENT element-name ANY>

To add the `<any>` data type, it is necessary to save all subsequent data of an entity, regardless of the names and values of this entity.

Note:
The <any> entities do not specify "real" entities; they are used as a container for the not parsed data and contain the entities with their entire content (attributes, etc.). An <any> entity is represented by a ##ANY variable of type (A) dynamic.

Because it can be necessary to access this data, a more specific name should be used instead of ##ANY followed by a generic number. It is recommended to add the name of the parent entity. See below for an example:

```
1 HTML
  2 BODY
    3 ATTRIBUTES_OF_BODY
      4 BGCOLOR          (A) DYNAMIC
      4 ##ANY_ATTR_BODY (A) DYNAMIC
        3 ##ANY_BODY         (A) DYNAMIC
```

If, during the parse process, an element of type <any> is found, all subsequent data is collected.

When serializing, all data is taken without changes and is added to the resulting XML document.

Restrictions for xs:any

Even if the attributes "maxOccurs" and/or "minOccurs" for xs:any are specified, the Natural variable implementing xs:any is always a scalar. The Natural variable may contain data of more then one entity.

The attribute "namespace" for xs:any is not evaluated, the Natural variable implementing xs:any may contain entities of different namespaces.

The attribute "processContents" for xs:any is not evaluated, because the parser used is not validating.

If a xs:choice or xs:sequence contains more then one definition of xs:any, the generation ends with an error, because during parse different <any> containers can not be recognized.

If a document contains entities that are not specified at the XML schema, and at the same level xs:any is defined, the Natural variable implementing xs:any may contain this "nonspecified" entity data.

Natural: Generation of an XML Schema or DTD with ##ANY Wildcards

During generation of an external data structure, each variable prefixed with ##ANY will be converted to the specific wildcard functions:

- ##ANY_ -> any entity type. Applies to DTDs and XML Schema.
- ##ANY_ATTR_ -> any attribute type. Applies to XML Schema only.