ABAP – XML Mapping

Peter McNulty,
NetWeaver Product Management
SAP Labs, LLC
## Intro

<table>
<thead>
<tr>
<th>XML in Open Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML in ABAP</td>
</tr>
<tr>
<td>ABAP – XML Mapping: Concepts</td>
</tr>
<tr>
<td>ABAP – XML Mapping with XSLT</td>
</tr>
<tr>
<td>Simple Transformations</td>
</tr>
<tr>
<td>Summary</td>
</tr>
</tbody>
</table>
The Web is evolving …

- … from **HTML** – a **fixed** language for **human viewing**
- … to **XML** – a **generic** language for **machine processing**

- **People** navigate through information in the form of **HTML** documents.
- **Machines** exchange **data and metadata** in the form of **XML** documents.
Example: Slashdot.org

RSS feed (news syndication)

URI in:

http://slashdot.org/it.rss

XML out:

```xml
<item rdf:about="http://it.slashdot.org/article.pl?sid=04/08/21/204210">
  <title>Microsoft Renovates Office Suite as a Web Service</title>
  <link>http://it.slashdot.org/article.pl?sid=04/08/21/204210</link>
  <description>fooblr writes "According to an article in EcommerceTimes, Microso
a product to an online service with a focus on automating collaborative work.
faster, easier and more efficient will be the next revolution in worker producti
forefront,' said Peter Rinearson, vice president for new business development
group"."</description>
  <dc:creator>michael</dc:creator>
  <dc:subject>microsoft</dc:subject>
  <dc:date>2004-08-22T02:32:00+00:00</dc:date>
  <slash:section>it</slash:section>
  <slash:department>you-will-be-assimilated</slash:department>
  <slash:comments>352</slash:comments>
  <slash:hitparade>352,343,264,208,76,52,35</slash:hitparade>
</item>
```

... more effective viewing with RSS aggregators
Example: Xignite Financial Web Services

Realtime currency exchange rates

URI in:

http://www.xignite.com/xCurrencies.asmx/GetRealTimeCrossRateTable?
Symbols=USD%2CEUR%2CJPY%2CGBP&InvokeButton=Invoke&
xMethod=GET

XML out:

```
<From>
  <Symbol>USD</Symbol>
  <Name>Dollar</Name>
</From>
<ExchangeRates>
  <ExchangeRate>
    <Outcome>Success</Outcome>
    <Delay>0</Delay>
    <To>
      <Symbol>EUR</Symbol>
      <Name>Euro</Name>
    </To>
    <Date>8/23/2004</Date>
    <Time>8:04:00 PM</Time>
    <Rate>0.823184996843338</Rate>
    <Text>1 Dollar = 0.82318 Euro</Text>
  </ExchangeRate>
</ExchangeRates>
```

Call a business service from ABAP ...

Or offer one.
Example: Amazon Web Services

Transactions on the Amazon marketplace

URI in:

http://xml.amazon.com/onca/xml3?locale=us&t=te&dev-t=te&KeywordSearch=ABAP&mode=books&sort=+daterank&offer=All&type=lite&page=1&f=xml

XML out:

```xml
<Details url="http://www.amazon.com/exec/obidos/ASIN/1592290126/link_code=xm2">
   <Asin>1592290116</Asin>
   <ProductName>ABAP Objects: The Official Reference</ProductName>
   <Catalog>Book</Catalog>
   <Authors>
      <Author>Horst Keller</Author>
      <Author>Joachim Jacobitz</Author>
   </Authors>
   <ReleaseDate>05 October, 2002</ReleaseDate>
   <Manufacturer>SAP Press</Manufacturer>
</Details>
```

Retrieval: GET  
Change: POST
From a programmer’s perspective, **XML** is ...

- a *generic data model* ("XML Infoset") for **trees** with
  - ordered, structured nodes ("elements")
  - unordered, unstructured nodes ("attributes")

- a *generic syntax* ("markup") for representation of trees

- the basis for an open set of **standards and tools**
  - *parsing*: text → tree;  *rendering*: tree → text
  - *typing, validating*: XSchema, RELAX NG
  - *querying, transforming*: XPath, XSLT

→ a *generic data structure concept* for programming languages
  - primitive: DOM, SAX
  - data binding: JAXB
Reasons for using XML

- handle document-like data (e.g. forms)
- represent data under heavy schema evolution
  - extensibility, flexibility, persistence
- combine heterogeneous data
  - "framework" applications, middleware, ...
- platform-independent modeling
  - "repositories", e.g. SAP WebDynpro
- ignorance / abuse
  - too lazy to design & implement adequate object model
- open communication format

...
Using XML

Problems solved

- standard syntax
- standard tools
- can model almost anything

Problems gained

- performance drain
- no semantics
- unclear relationship with data structures (data conversion)
  - Part II (XML from/to ABAP)
- schema inflation
  - "Business ML" by standards body X
  - "Business ML" by vendor Y
  - Proprietary format by application Z

need for transformations
Tree transformations with XSLT

(a) XML tree to XML tree (data-centric)
(b) XML tree to text (document-centric)
**Transforming XML with XSLT (2)**

**XSLT** is ...
- a high-level *tree-transformation* language
- with XML syntax
- declarative
  - no "state"
- rule-based
  - pattern matching against source tree
- functional
  - source tree navigation by XPath expressions
- compositional
  - result tree construction by XSLT instructions mixed with literal XML fragments
- "the SQL of the Web"

* "Extensible Stylesheet Language / Transformations"
**XPath example: tree navigation**

```
book[editor[name/last='Smith' and name/first='John']]
//section[contains(@head,'DOM')]
/ancestor::chapter[1]/author
```
Intro

XML in Open Systems

**XML in ABAP**

ABAP – XML Mapping: Concepts

ABAP – XML Mapping with XSLT

Simple Transformations

Summary
The iXML package (since 4.6D): features

- implemented in kernel (C++)
- encapsulated in ABAP proxy classes
- general XML parser / renderer
- event-based parser (~ SAX)
- DOM (Document Object Model)
  - superset of DOM level 1 (incl. XML namespaces)
- Validation
  - DTD (since 6.10)
  - not XMLSchema
**iXML interfaces in ABAP Objects**

- **OO design:** interfaces, inheritance, factories
- **main factory class** `CL_IXML`
- **~40 Interfaces**
  - `IF_IXML_PARSER`, `IF_IXML_NODE`, ...
- **documentation:** SAP Library
  - [→ “SAP NetWeaver Components” ]
  - → “SAP Web Application Server”
  - → “Basis Services / Communication”
  - → “XML Library”
- **used by many SAP applications**
iXML example programs in package SIXML_TEST

- parsing into a DOM: T_PARSING_DOM
- manipulating a DOM: T_DOM_MANIPULATE
- rendering a DOM: T_RENDERING_DOM
- validating with a DTD: T_DTD_VALIDATION

data element type ref to if_ixml_element.

element = document->create_element( name = 'date' ).
element->set_attribute( name = 'format' value = 'yyyyymmdd' ).
element->set_value( value = ldate ).
XML in ABAP: XSLT Processor (1)

The SAP XSLT processor (since 6.10): features

- performance implementation in SAP kernel (C++)
- scalability optimization for server-side execution
- interoperability with iXML package
- conformance (except for justified omissions)
- integration into language environment call from ABAP, call back to ABAP
- integration into development environment workbench, transport
XML in ABAP: XSLT Processor (2)

Unimplemented XSLT 1.0 features
- xsl:number
- Forwards-compatible processing, fallback processing
- Attribute sets
- Namespace aliasing

Implemented XSLT 2.0 features
- Grouping (xsl:for-each-group)
- User-defined XPath functions (xsl:function)
- Multiple results (xsl:result-document)

- ABAP Calls (by extension function or extension instruction)
- many extension functions
- XPath 2.0 features
Extension example: **ABAP Call** by extension function from XPath

**At top level:** Declare external function

```xml
< sap:external-function name="p:f1" method="METH">
  < sap:argument param="IP_1"/>
  < sap:argument param="IP_2"/>
  < sap:result param="EP" type="number"/>
</ sap:external-function>
```

**Function parameter declarations**

**Function result declaration**

**In XPath expression:** Invoke instance method on external object

```xml
<xsl:value-of select="p:f1($obj, 42, 'foo') + 1="/>
```

**XSLT variable bound to ABAP object**

**XPath parameter values**
XML in ABAP: Invocation of XSLT

ABAP statement: CALL TRANSACTION

TRY.
  CALL TRANSACTION my_trans
  or (my_trans_name)

  PARAMETERS p_1 = my_par_1 ... p_n = my_par_n
  or (my_par_table)

  SOURCE XML my_xml_source

  RESULT XML my_xml_result.

CATCH cx_transformation_error INTO exc.
  ...
ENDTRY.
XML in ABAP: XSLT Development

Workbench integration

- **SE80:**  
  
  *Edit object → More... → Transformation*

- object tree, context menu:  
  
  *Create → More... → Transformation*

- direct: transaction **STRANS (6.20: XSLT_TOOL)**

- testing: transaction **XSLT**

- programs must be *activated* before use

- check / activate triggers compilation

<table>
<thead>
<tr>
<th>XSLT Program SXSLTDEMO_FLIGHTS_CONNECTIONS</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>unknown variable 'modes'</td>
<td></td>
</tr>
<tr>
<td>XSLT Program SXSLTDEMO_FLIGHTS_CONNECTIONS</td>
<td>20</td>
</tr>
<tr>
<td>unsupported instruction 'xsl:sorts'</td>
<td></td>
</tr>
</tbody>
</table>

- maintenance API: function **XSLT_MAINTENANCE**
Demo

- Transformations in the ABAP Workbench
  Check – Activate – Test
- “Flights” example (SSTDEMO2):
  - XML to XML with XSLT
  - XML to HTML with XSLT
<table>
<thead>
<tr>
<th>Intro</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML in Open Systems</td>
</tr>
<tr>
<td>XML in ABAP</td>
</tr>
<tr>
<td><strong>ABAP – XML Mapping: Concepts</strong></td>
</tr>
<tr>
<td>ABAP – XML Mapping with XSLT</td>
</tr>
<tr>
<td>Simple Transformations</td>
</tr>
<tr>
<td>Summary</td>
</tr>
</tbody>
</table>
XML ↔ ABAP: Application Areas

Who needs (de)serialization?

- XML-based communication middleware
  - SAP Exchange Infrastructure (XI)
  - Web Services

- XML-based persistence
  - SAP Archiving

- XML repositories ↔ DB tables import / export
  - SAP WebDynpro metadata

- application-specific uses
  - XML via HTTP / in database / ...
ABAP: Structure Mapping

The **structure mapping** problem

- **external applications**
- **external XML format**
- **structure difference**
- **XML-based communication**
- **ABAP applications**
- **ABAP data structure**
- **ABAP functionality**
Which side is driving?

- **inside-out** approach
- **outside-in** approach
- **symmetric** approach (6.20)
  - canonical XML encoding of data structures
  - transformation with XSLT
- **symmetric** approach (6.40)
  - dedicated XML / data transformation language
  - no conceptual "canonical encoding" indirection
The **inside-out** approach

1. **inside**
   - ABAP functionality
   - 6.20 kernel

2. **outside**
   - ABAP data structure
   - XML handler

3. **inside**
   - ABAP functional mapping (XSLT...)
   - external XML format
   - canonical XML encoding

© SAP AG 2004, SAP TechEd / ABAP 252 / 29
The outside-in approach ("data binding")

1a. generated

2. generated

3. [generated]

6.20 kernel

XML handler

ABAP proxy

ABAP proxy interface

ABAP adapter

ABAP data structure

ABAP functionality

external XML format
The symmetric approach (6.20)

1. ABAP functionality
2. ABAP data structure

1. XSLT programs
2. Mapping Engine (6.20 Kernel)

external XML format
Outbound XML

- XML
- XSLT (outbound)
- Transform DOM to outbound XML
- Construct canonical DOM for data tree

ABAP → XML with XSLT
Inbound XML

- XML
  - parse inbound XML to DOM
  - XSLT (inbound)
  - conceptual: transform to canonical XML tree
  - transform DOM to data tree (directly)
  - ABAP
Scope:

- **not a** communication mechanism
- **not an** RPC framework
- **only** a mapping engine
Intro
XML in Open Systems
XML in ABAP
ABAP – XML Mapping: Concepts
**ABAP – XML Mapping with XSLT**
Simple Transformations
Summary
### Serialization

```
CALL TRANSFORMATION ...  
  PARAMETERS ... 
  OPTIONS option_1 = string ... option_n = string 
  SOURCE XP_1 = my_var_1 ... XP_n = my_var_n  
    or (my_var_table) 
  RESULT XML my_xml_result. 
```

### Deserialization

```
CALL TRANSFORMATION ...  
  PARAMETERS ... 
  OPTIONS option_1 = string ... option_n = string 
  SOURCE XML my_xml_source 
  RESULT XP_1 = my_var_1 ... XP_n = my_var_n  
    or (my_var_table). 
```
ABAP → XML: Invocation of XSLT (Example)

ABAP fragment

```abap
DATA num TYPE P LENGTH 5 DECIMALS 2.
DATA txt TYPE STRING.
num = '1.23-'.
 txt = 'Yes, 2 < 3'.
CALL TRANSFORMATION id
   SOURCE Foo = num  bar = txt
   RESULT XML  my_xml_result.
```

XML result fragment

```xml
... <FOO>-1.23</FOO>
<BAR>Yes, 2 &lt; 3</BAR>
... 
```

- **Identity transformation**: symbolic names are normalized to uppercase.
- **Format depends on corresponding ABAP variable**: format depends on corresponding ABAP variable.
XML source fragment

...  
<FOO>-1.23</FOO>
<BAR>Yes, 2 &lt; 3</BAR>
...

ABAP fragment

DATA num TYPE P LENGTH 5 DECIMALS 2.
DATA txt TYPE STRING.
CALL TRANFORMATION id
  SOURCE XML my_xml_source
  RESULT Foo = num bar = txt.
WRITE: / num, / txt.

1.23-
Yes, 2 < 3
ABAP Types and the asXML Format

- ABAP → asXML : type of ABAP source variable(s) determines generated asXML representation
- asXML → ABAP : type of ABAP result variable(s) determines required asXML representation

→ source / result ABAP variables must be typed

A Data Exchange Format

- XML-Schema datatypes
- "Human-readability" not a top priority
  - no dependence on customization
  - no dependence on locale settings
  - no use of conversion exits
### Simple Types

<table>
<thead>
<tr>
<th>ABAP Type</th>
<th>ABAP Example*</th>
<th>XML Schema Type</th>
<th>XML Example</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING</td>
<td>□Hi□□</td>
<td>string</td>
<td>□Hi□□</td>
<td>string of characters</td>
</tr>
<tr>
<td>C</td>
<td>□Hello</td>
<td>string</td>
<td>□Hello</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>001234</td>
<td>string ([0-9]+)</td>
<td>001234</td>
<td>string of digits</td>
</tr>
<tr>
<td>I</td>
<td>123–</td>
<td>int unsignedByte, short</td>
<td>−123</td>
<td>number</td>
</tr>
<tr>
<td>P</td>
<td>1.23–</td>
<td>decimal</td>
<td>−1.23</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>−3.14E−2</td>
<td>double</td>
<td>−3.14E−2</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>20010820</td>
<td>date</td>
<td>2001-08-20</td>
<td>ISO 8601 date/time</td>
</tr>
<tr>
<td>T</td>
<td>201501</td>
<td>time</td>
<td>20:15:01</td>
<td></td>
</tr>
<tr>
<td>XSTRING</td>
<td>456789AB</td>
<td>base64Binary</td>
<td>RWeJqw==</td>
<td>base64-encoded binary data</td>
</tr>
<tr>
<td>X</td>
<td>ABCDEF</td>
<td>base64Binary</td>
<td>q83v</td>
<td></td>
</tr>
</tbody>
</table>

* Internal string representation of ABAP datatypes
asXML Format: Structures

**Structures**

```abap
DATA miles TYPE person.
miles-name = 'Miles Davis'.
miles-born = '19260526'.
CALL TRANSACTION id
  SOURCE     LEGEND = miles
  RESULT XML my_xml_result.
```

DDict Structure **PERSON:**
- **NAME** TYPE STRING
- **BORN** TYPE D

... `<LEGEND>`
  `<NAME>Miles Davis</NAME>`
  `<BORN>1926-05-26</BORN>`
...`

Component names determine XML sub-element names.
Internal Tables

```abap
DATA: one TYPE person.
DATA: itab TYPE STANDARD TABLE OF person.

one-name = 'John Coltrane'. one-born = '19260923'. APPEND one TO itab.
one-name = 'Miles Davis'. one-born = '19260526'. APPEND one TO itab.
one-name = 'Charlie Parker'. one-born = '19200829'. APPEND one TO itab.

CALL TRANSFORMATION id
  SOURCE GREATEST = itab
  RESULT XML my_xml_result.

...<GREATEST>
  <PERSON><NAME>John Coltrane</NAME> <BORN>1926-09-23</BORN></PERSON>
  <PERSON><NAME>Miles Davis</NAME> <BORN>1926-05-26</BORN></PERSON>
  <PERSON><NAME>Charlie Parker</NAME><BORN>1920-08-29</BORN></PERSON>
</GREATEST>
...```

line type from DDict (or `<item>`)
<asx:abap version = "1.0"
xmlns:asx = "http://www.sap.com/abapxml">
  <asx:values>
    <XP_1> ... </XP_1>
    ... 
    <XP_n> ... </XP_n>
  </asx:values>
</asx:abap>

---
additional XML elements for objects
and referenced data structures

---

list of ABAP-XML bindings
Advantages of the XSLT solution

- **no restrictions**
  - arbitrary XML schemas
  - arbitrary data & object types
    - graphs of objects
  - arbitrarily complex structural transformations

- **no redundancy**
  - no generation of schemas from types (schema inflation)
  - no generation of types from schemas (type inflation)
  - each side retains its structures

- **high abstraction level**
  - no low-level XML handling in applications
  - separate, expressive transformation language
Disadvantages of the XSLT solution

- **learning XSLT**
  - overkill for simple conversion tasks
  - no tool support

- **asymmetric programs**
  - one for XML → ABAP, one for ABAP → XML

- **resource consumption (Time & Space)**
  - \( T / S \) : DOM construction (on source side)
  - \( T \) : codepage conversions (internal encoding ≠ SAP CP)
  - \( T \) : XSLT engine overhead (complex state, powerful operations)

- **no static type checking**
Intro
XML in Open Systems
XML in ABAP
ABAP – XML Mapping: Concepts
ABAP – XML Mapping with XSLT
Simple Transformations
Summary
XML ↔ ABAP without XSLT: Requirements

Requirements for a dedicated XML/ABAP mapping engine

- **time**: increase throughput by factor $\geq 10$
- **space**: increase / eliminate upper limit on size of data
- **ease of use**
  - simple syntax & semantics, statically type-checked
  - tool support for creating mappings
  - one program for XML→ABAP and ABAP→XML

Deliberate trade-offs

- lower expressive power (but cover 90% of typical applications)
- not usable for XML ↔XML

⇒ "Simple Transformations"
### Simple Transformations: Tree Access

#### Tree access in XSLT vs. Simple Transformations

<table>
<thead>
<tr>
<th>Serialization</th>
<th>XSLT</th>
<th>Simple Transformations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ABAP source</strong></td>
<td>random access (XPath on canonical DOM)</td>
<td>random access</td>
</tr>
<tr>
<td><strong>tree navigation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>XML result</strong></td>
<td>linear</td>
<td>linear</td>
</tr>
<tr>
<td><strong>tree construction</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deserialization</th>
<th>XSLT</th>
<th>Simple Transformations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XML source</strong></td>
<td>random access (XPath)</td>
<td>linear</td>
</tr>
<tr>
<td><strong>tree navigation</strong></td>
<td></td>
<td>(stream reader)</td>
</tr>
<tr>
<td><strong>ABAP result</strong></td>
<td>linear (modulo component order)</td>
<td>random access</td>
</tr>
<tr>
<td><strong>tree construction</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Simple Transformations: Expressive Power

Anything that can be done with ...

- accessing each node in the data tree any number of times
- accessing each node in the XML tree at most once, in document order (with "lookahead 1" on XML source)

... which includes (any combination of) ...

- renamings (e.g.: structure-component / element names)
- projections (omission of sub-trees)
- permutations (changes in sub-tree order)
- constants (e.g.: constant values, insertion of tree levels)
- defaults (for initial / special values)
- conditionals (e.g.: existence of sub-trees, value of nodes)
- value maps

→ covers most data mappings in practice
Simple Transformations: Key Features (1)

Programs are XML templates

- literal XML with interspersed instructions
- declarative, straightforward semantics

Data tree access by node references

- instructions access data by simple “reference expressions”
- all named children of a data node are accessible by name
- tables are accessible as a whole (all lines or none)

```xml
<Customer>
  <tt:loop ref="CUSTTAB">
    <LastName>
      <tt:value ref="NAME.LAST"/>
    </LastName>
  </tt:loop>
</Customers>
```
Simple Transformations: Key Features (2)

Programs are reversible

- **serialization** : write (produce) tokens to stream
- **deserialization** : match (consume) tokens from stream
- invocation determines direction
- if no asymmetric construct is used:
  
  - \( D[program] ( S[program] (data) ) = data \)
  - \( S[program] ( D[program] (document) ) \approx document \)

(D: deserialization, S: serialization)
“XSLT” is generalized to:

“Transformation” = “XSLT” or “Simple Transformation”

- workbench integration
  - same workbench / transport object type
  - same access paths
  - transformation type determined in creation dialog
  - `XSLT_MAINTENANCE` works for both types

- ABAP integration
  - `CALL TRANSFORMATION` works for both types
  - both types take same XML input / output types (e.g. iXML streams)
  - exception hierarchy generalized (new common root)

→ uniform appearance (easy to switch from XSLT solutions)
Simple Transformations: Implementation

Virtual machine (VM)
- compilation of programs to bytecode
- storage of bytecode in database / buffering in shared memory
- interpretation of bytecode in kernel (lean engine)
→ scalable performance

XML stream reader / writer
- reader: efficient "token pull" discipline
- specialized for standard encoding UTF-8
- also usable directly from ABAP: classes CL_FX_READER, CL_FX_WRITER
→ limit on data size lifted
“Flights” example (SSTDEMO2):

- ABAP to XML with ST
- XML to ABAP with ST
Simple Transformations: Programs

```xml
<tt:transform
xmlns:tt="http://www.sap.com/transformation-templates">
  ( <tt:include name="name" /> )* 

  ( <tt:root name="name" /> )* 

  ( variable / parameter declaration ) *

  ( <tt:template [name="name"]>
    [ context declaration ]
    template content
    </tt:template> )* 

</tt:transform>
```
**ST Constructs: Value**

**Value**

<table>
<thead>
<tr>
<th>S</th>
<th>ref-node value to XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>XML value to ref-node</td>
</tr>
</tbody>
</table>

Example copy value from/to field CHAR, with special mappings:

<table>
<thead>
<tr>
<th>S</th>
<th>map {*, +, –} to ~</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>map ~ to *</td>
</tr>
</tbody>
</table>

```xml
<tt:value
    ref="CHAR"
    map=""
    val('*') = xml('~'),
    val('+','-') > xml('~")
/>```
ST Constructs: Literal Element / Attribute

Literal element / attribute

S write literal content

D match literal content (no match → error)

example

```xml
<Time zone="CET" tt:ref="TIME">
  <tt:value/>
</Time>
```

alternative

```xml
<Time zone="CET" tt:value-ref="TIME"/>
```

XML

```xml
<Time zone="CET">12:59:00</Time>
```
ST Constructs: Literal Text

Literal text

S write literal text

D match literal text

example

```
<Time>
  <tt:value ref="TIME"/>
  <tt:text> CET</tt:text>
</Time>
```

alternative

```
<Time>
  <tt:value ref="TIME"/>
  CET
</Time>
```

XML

```
<Time>12:59:00 CET</Time>
```
**ST Constructs: Attribute**

**Attribute**

<table>
<thead>
<tr>
<th>S</th>
<th>write attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>match attribute</td>
</tr>
</tbody>
</table>

**Example**

```xml
<Time tt:ref="ZTIME">
  <tt:attribute name="zone" value-ref="TZONE"/>
  <tt:value ref="TIME"/>
</Time>
```

```xml
<Time zone="CET">12:59:00</Time>
```
ST Constructs: Deep Copy

Deep Copy

S  copy sub-tree below ref-node to stream

D  copy sub-tree below current XML node to ref-node

example

XML

<tt:copy ref="ZTIME_TAB"/>

<ZTIME>
  <TZONE>CET</TZONE>
  <TIME>12:59:00</TIME>
</ZTIME>

<ZTIME>
  <TZONE>EST</TZONE>
  <TIME>06:59:00</TIME>
</ZTIME>
ST Constructs: Table Loop

Table Loop

S  evaluate content for all lines of table

D  evaluate content, insert lines into table until matching fails

example

<tt:loop ref="ZTIME_TAB"/>
  <tt:copy/>
</tt:loop>

XML

<TZONE>CET</TZONE>
<TIME>12:59:00</TIME>
<TZONE>EST</TZONE>
<TIME>06:59:00</TIME>
**Basic Conditional**

**S** [if data condition is true:] evaluate template content

**D** [if template content matches:] evaluate template content [establish assertions] [evaluate check-clause]

**example**

**S** if \( \text{TZONEN}= 'CET' \) and \( \text{TIME} = t > 000000 \):
write \( <\text{CET} \rangle t <\text{/CET} \rangle \)

**D** if \( <\text{CET} \rangle t <\text{/CET} \rangle :\)
\( \text{TIME} := t, \text{TZONEN} := 'CET', \)
check \( \text{TIME} > 000000 \)
Value Assertions

ref-nodes in expressions

ref :
• general: ref('name')
• simplified: name

for name of the form
letter ( letter | digit | _ )*  

example

COUNT = 1,
ref('PRICE.%OFF') = P('7.5')

<tt:cond data = " assertion ( , assertion )*">

• initial(ref)
• ref = constant

constants

• C('string') or 'string'
• D('yyyyymmdd')
• F(float)
• I(integer) or integer
• N('digits')
• P('decimal')
• T('hhmmss')
• X('xstring')
ST Constructs: Basic Conditional (3)

Check-clause

```
<tt:cond [s-|d-] check="expr"/>
```

```
expr:

• ref op (constant | ref) with op in {=, !=, <, >, <=, >=}

• initial(ref), not-initial(ref)

• exist(ref) (node-existence check)

• t-op(ref) with t-op = type-C|D|F|I|N|P|T|X (node-type check)

• expr and expr

• expr or expr

• not(expr)

• (expr)

Example

TIME>CTIME and (initial(TZONE) or TZONE='CET')
```
ST Constructs: Composite Conditional

Switch

**S** evaluate first case with true data condition

**D** evaluate first case with matching pattern

```
<tt:switch>
  case *
</tt:switch>
```

```
<tt:group>
  case :
  • <tt:cond ...>
  • <tt:s-cond ...> (only serialization)
  • <tt:d-cond ...> (only deserialization)
</tt:group>
```

Group

**S** evaluate all cases with true data condition

**D** evaluate all cases with matching pattern

```
<tt:group>
  case *
</tt:group>
```
**Variable / Parameter Declaration**

**declaration**:

- `<tt:root name="name"/>
- `<tt:variable name="name" [val="constant"]/>
- `<tt:parameter name="name" [val="constant"] [kind="in" | "out" | "in/out"]/>

**Context of main template**: top-level declarations

**Default context of sub-template**: one unnamed root

**initial / default value in val**

**parameter direction (kind) independent of transformation direction**
Apply Template

```xml
<tt:apply
  name="template-name">
  binding*
</tt:apply>

binding :
  • <tt:with-root name="name"
    [ref="ref"] />
  • <tt:with-parameter name="name"
    [ref="ref"
    | var="name"
    | val="constant"] />
```

Call Transformation

- `tt:apply`: named template in main or included program
- `tt:call`: existing ST program (separate load)
ST Constructs: Variable [De]Serialization

Read Variable

S  no effect
D  XML value to variable

Write Variable

S  variable value to XML
D  no effect

Example XML code:

```xml
<tt:read var="name"
    [type="C|D|F|I|N|P|T|X"]
    [map="mapping-list"] />
```

```xml
<tt:write var="name"
    [map="mapping-list"] />
```
### ST Constructs: Assignment

#### Assign to Ref-Node

**S**  no effect

**D**  assign to ref-node

```xml
<tt:assign to-ref="ref"  
( var="name" | val="constant" ) />
```

```xml
<tt:clear ref="#ref" />
```

#### Assign to Variable

**S**  assign to variable

**D**  ... except if ref (no effect)

```xml
<tt:assign to-var="name"  
[ var="name" | ref="#ref"  
| val="constant" ] />
```

```xml
<tt:clear var="#name" />
```

**example**

same effect as

```xml
<tt:value ref="#A" />
```

```xml
<tt:assign to-var="#V" ref="#A"/>
<tt:write var="#V"/>
<tt:read var="#V"/>
<tt:assign to-ref="#A" var="#V"/>
```

```xml
<tt:assign to-ref="#A" var="#V"/>
```
**ST Constructs: Variable Conditional**

### Basic Variable Condition

```html
<tt:cond-var check="check-clause"
template content
</tt:cond-var>
```

### Variable Switch

```html
<tt:switch-var>
case*
</tt:switch-var>
```

- **check-clause** uses variables (not ref-nodes)
- At most one case in `tt:switch-var` may have no condition (default)
ST Constructs: Directional

Serialize
S  evaluate template content
D  no effect

 Deserialize
S  no effect
D  evaluate template content

Skip
S  no effect
D  match & skip [number of] nodes [named element-name]

<tt:serialize>
  template content
</tt:serialize>

<tt:deserialize>
  template content
</tt:deserialize>

<tt:skip [name="element-name"]
  [count="number" | "*"]>
  template content
</tt:skip>
ST Exception Hierarchy

- CX_TRANSFORMATION_ERROR
  - CX_ST_ERROR
    - CX_ST_REF_ACCESS
    - CX_ST_CONDITION
      - CX_ST_COND_CHECK_FAIL
      - CX_ST_GROUP_MISSING_CASE
      - CX_ST_SWITCH_NO_CASE
    - CX_ST_MATCH
      - CX_ST_MATCH_NAMED
        - CX_ST_MATCH_ATTRIBUTE
        - CX_ST_MATCH_ELEMENT
      - CX_ST_MATCH_TEXT
      - CX_ST_MATCH_TYPE
  - CX_ST_FORMAT_ERROR
  - CX_ST_DESERIALIZATION_ERROR
  - CX_ST_SERIALIZATION_ERROR
  - CX_ST_INVALID_XML
  - CX_ST_CALL_ERROR
Summary: ABAP – XML Mapping

ABAP Data → Simple Transformations → XML Doc

New in 6.40

HTML / Text → XSLT → DB

Network

© SAP AG 2004, SAP TechEd / ABAP 252 / 74
Summary: Available Demos

Transactions in the Basis delivery

- (6.20) **SXSLTDEMO1** : XSLT

- (6.40) **SSTDEMO1** : ST

- (6.40 SP9) **SSTDEMO2** : XSLT and ST
Summary: Transformation Languages for ABAP

**XSLT** (since 6.10)
- works on canonical XML representation of ABAP data (asXML)
- builds DOM for source side
- arbitrarily complex transformations

**Simple Transformations** (since 6.40)
- only for ABAP ↔ XML
- only linear transformations (no DOM)
- speedup over XSLT: 10 – 30; “unlimited” size of data
- reversible (one program for both directions)

Both
- symmetric: no generation of ABAP code / XML schemas
- integrated in workbench (maintenance / transport)
- integrated in ABAP: `CALL TRANSFORMATION`
Summary: When To Use What

System Landscape Integration
  ➔ Exchange Infrastructure

RPC-Style Web Services
  ➔ ABAP Web Services

Direct XML Processing in ABAP
  ▪ REST-Style Web Services (URI in, XML out)
  ▪ Custom XML Persistence
  ▪ XML-Based Repositories
  ▪ ...

Simple mappings, high throughput
  ➔ Simple Transformations
Complex mappings, limited throughput
  ➔ XSLT
Questions?

Q&A
No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, and Informix are trademarks or registered trademarks of IBM Corporation in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

MaxDB is a trademark of MySQL AB, Sweden.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.