

European Tivoli Technical Conference



Tivoli. software

4-8 October 2010
Dresden
Germany

Session Number: SEC04

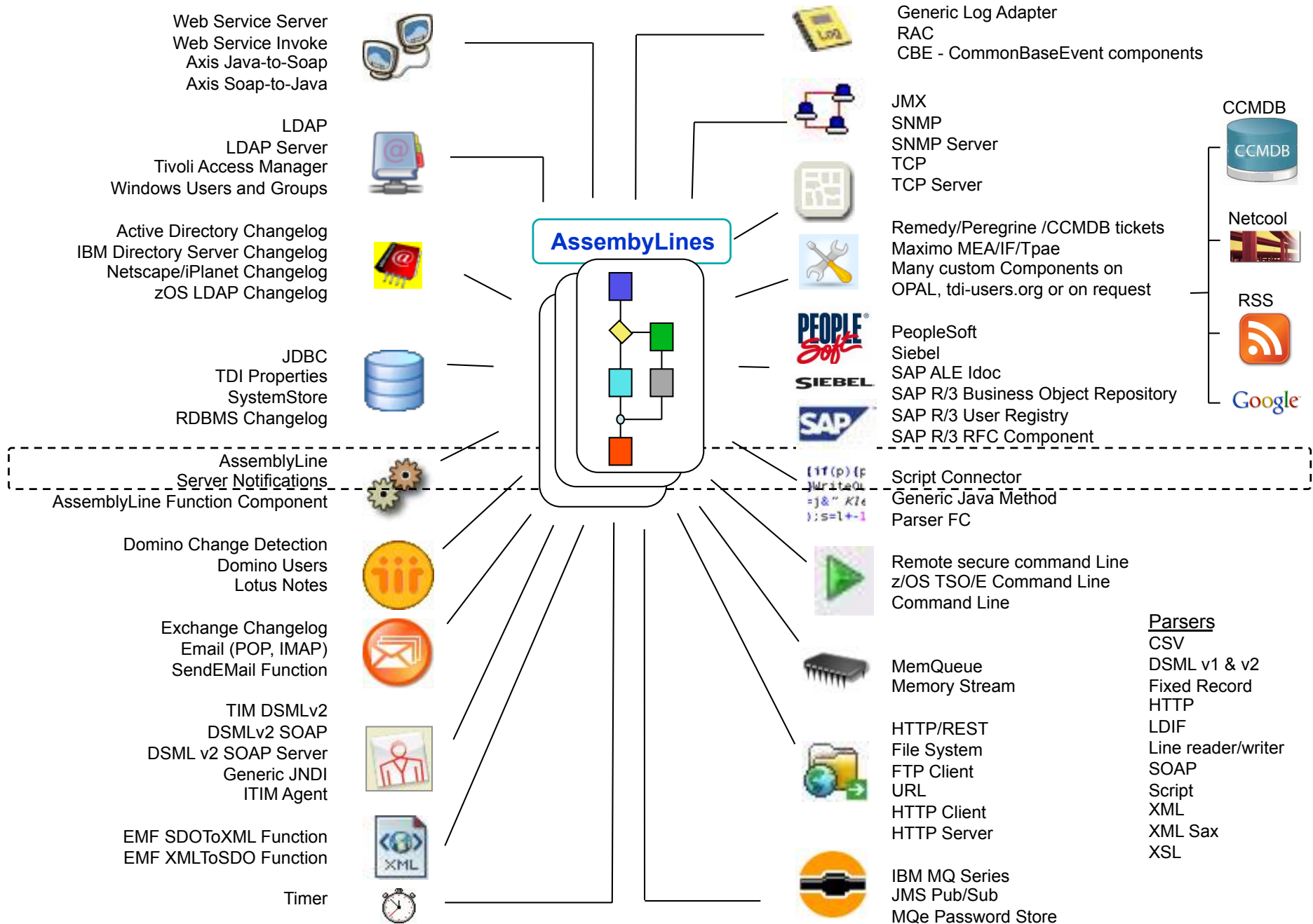
Presentation Title: (Very) Advanced XML transformation with IBM Tivoli Directory Integrator

Speaker: Franz Wolfhagen, IBM Denmark

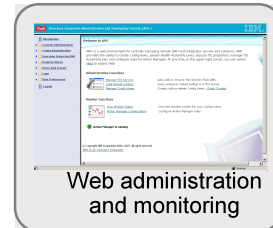
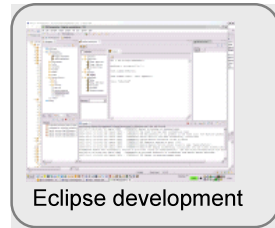
Designation: Advanced

Why use ITDI

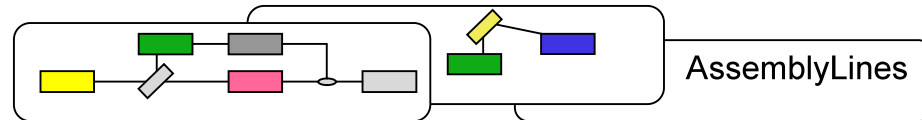
The "Gaffer Tape" of IT Integration



Human interfaces



Workflow



Transformation

Functions	JavaScript	Attribute maps	External Java libraries
-----------	------------	----------------	-------------------------

Services

Change detection	Connector pooling	Logging & tracing	Persistence in memory, rdbms, message queues
------------------	-------------------	-------------------	--

Parsers

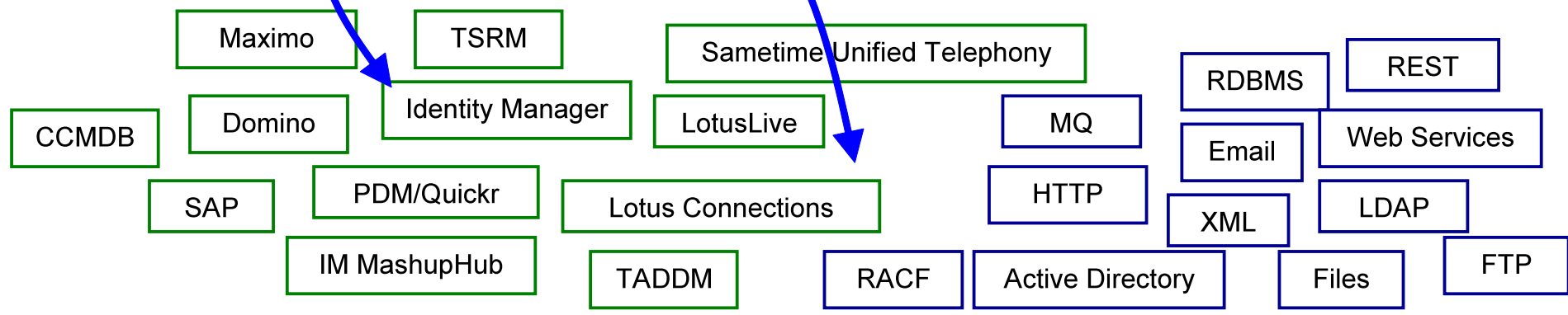
Out-of the box	Plug-in framework for JavaScript and Java
Out-of the box	

Connectors

Out-of the box

A P I	RMI
	JMX
	REST

Platforms



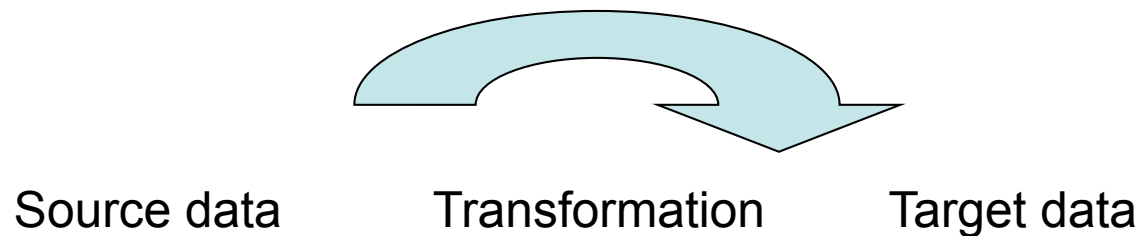
ITDI and XML

XML Transforms

Getting from here to there

The Basic Transformation

- Going from some source data
- Transform (manipulate) the data into some target format



- Either source or target will be XML here...

Examples of Transformation

Source Data	Transformation	Target Data
XML File	XML Parser	ITDI Work bucket
ITDI Work Bucket	XML Parser	XML File
ITDI Entry	ITDI entry2XML()	XML string
XML and XSL strings	ITDI xslTransform	(X)HTML string

- **Countless possibilities....**

ITDI XML Transformations methods

- Using the simple XML Parser
- Using the (StAX) XML Parser
- Using the XSL based XML Parser
- Using the XML utilities
- Using the `Entry.toXML()`
- Using the `XML.toXML()` and `XML.fromXML()`
- Using Java methods

Using a "scripted" Parser

```
//Setup the parser from the Parsers Ressource
var myParser = system.getParser("Parsers/someParser");

//Check if it worked
if (myParser == null) throw "Unable to get Parsers/someParser";

//Create a Java Outputstream and connect it the parser
os = new java.io.ByteArrayOutputStream();
myParser.setOutputStream(os);


//Initialize the parser
myParser.initParser();

//Write an entry to the parser and close it
myParser.writeEntry(someEntry);
myParser.closeParser();

//The parsed entry is now available in the outputstream
task.logmsg("Result : " + os.toString("UTF-8"));
```

Converting an Entry to XML

The Work Entry

 **createWork**

Inherit From

Map

Add

Delete

More...

Work Attribute	← Assignment
attr1	ret.value = "Single value";
attr2	ret.value = ["value1", "value2"];
æøå	ret.value = ["æøå", "ÆØÅ"];

Converting an Entry to XML Using the simple XML Parser

Simple XML Parser

Select Parser Help

Root Tag * DocRoot

Entry Tag * Entry

Value Tag * ValueTag

Comment

Detailed Log

Advanced

Character Encoding UTF-8

Omit XML Declaration

Document Validation

Namespace Aware

Indent Output

Converting an Entry to XML Using the simple XML Parser

```
var myEntry = work;

//Setup the XML parser
var xml = system.getParser("Parsers/xml");
if (xml == null) throw "Unable to get Parsers/xml";
os = new java.io.ByteArrayOutputStream();
xml.setOutputStream(os);
xml.initParser();
xml.writeEntry(myEntry);
xml.closeParser();

task.logmsg("This is the work entry as parsed with
the XML (simple) parser : \n" + os.toString
("UTF-8").trim() + "\n");
```

```
INFO - This is the work entry as parsed with the XML
      (simple) parser :
<DocRoot>
  <Entry>
    <attr2>
      <ValueTag>value1</ValueTag>
      <ValueTag>value2</ValueTag>
    </attr2>
    <attr1>Single value</attr1>
    <æøå>
      <ValueTag>æøå</ValueTag>
      <ValueTag>EØÅ</ValueTag>
    </æøå>
  </Entry>
</DocRoot>
```

Converting an Entry to XML Using the (StAX) XML Parser

XML Parser

Select Parser Help

Simple XPath	*
Entry Tag	Entry
Value Tag	ValueTag
Comment	

Detailed Log

Advanced

Prefix To Namespace Map	prefix=namespace
XSD Schema Location	
Character Encoding	UTF-8
Static Attribute Declarations	<!-- this is an example for statically declared XML attributes/namespaces --> <!-- DocRoot xmlns="defaultNS" attr1="val2" --> <Entry xmlns:p1="p1NS" p1:attr2="val2" /> </DocRoot -->

Ignore repeating XML declarations while reading

Converting an Entry to XML Using the (StAX) XML Parser

```
var myEntry = work;

//Setup the StAX XML parser

var xml2 = system.getParser("Parsers/XML2");

if (xml2 == null) throw "Unable to get Parsers/XML2";

os = new java.io.ByteArrayOutputStream();

xml2.setOutputStream(os);

xml2.initParser();

xml2.writeEntry(myEntry);

xml2.closeParser();

task.logmsg("This is the work entry as parsed with the
(StAX) XML parser : \n" + os.toString("UTF-8").trim
() + "\n");
```

```
INFO - This is the work entry as parsed with the (StAX)
XML parser :

<DocRoot>

  <Entry>

    <attr2>

      <ValueTag>value1</ValueTag>

      <ValueTag>value2</ValueTag>

    </attr2>

    <attr1>Single value</attr1>

    <æøå>

      <ValueTag>æøå</ValueTag>

      <ValueTag>EØÅ</ValueTag>

    </æøå>

  </Entry>

</DocRoot>
```

Converting an Entry to XML Using the XSL Parser

XSL based XML Parser

Select Parser Help

Comment

Detailed Log

▶ **Advanced**

▶ **Input**

▼ **Output**

Use output XSL file

Output XSL File Name

Output XSL

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:strip-space elements="*" />
  <xsl:output indent="yes" method="xml" omit-xml-declaration="yes" encoding="UTF-8"/>

  <xsl:template match="/">
    <xsl:copy-of select="." />
  </xsl:template>
```

Converting an Entry to XML Using the XSL Parser

```
var myEntry = work;

//Setup the xsl parser

var xsl = system.getParser("Parsers/xsl");

if (xsl == null) throw "Unable to get Parsers/xsl";

os = new java.io.ByteArrayOutputStream();

xsl.setOutputStream(os);

xsl.initParser();

xsl.writeEntry(myEntry);

xsl.closeParser();

task.logmsg("This is the work entry as parsed with
the xsl parser : \n" + os.toString("UTF-8").trim
() + "\n");
```

```
INFO - This is the work entry as parsed with the xsl
parser :

<?xml version="1.0" encoding="UTF-8"?>
<DocRoot>
  <Entry>
    <Attribute name="attr2">
      <Value>value1</Value>
      <Value>value2</Value>
    </Attribute>
    <Attribute name="attr1">
      <Value>Single value</Value>
    </Attribute>
    <Attribute name="æå">
      <Value>æå</Value>
      <Value>ÆÅ</Value>
    </Attribute>
  </Entry>
</DocRoot>
```

Converting an Entry to XML Using the `Entry.toXML()` method

```
var myEntry = work;
task.logmsg("This is the work entry as converted
with Entry.toXML() : \n" + myEntry.toXML().trim
() + "\n");
```

```
INFO - This is the work entry as converted with
Entry.toXML() :
<attr1>Single value</attr1>
<attr2>
value1
value2
</attr2>
<æøå>
æøå
EØÅ
</æøå>
```

Converting an Entry to XML Using the XML.toXML() method

```
myXML = new com.ibm.di.eclipse.http.XML()
var myEntry = work;

task.logmsg("This is the work entry as converted with
XML.toXML() : \n" + myXML.toXML(myEntry).trim() +
"\n");
```

```
INFO - This is the work entry as converted with
XML.toXML() :
```

```
<Entry>
  <Attribute name="attr1">
    <value>Single value</value>
  </Attribute>
  <Attribute name="attr2">
    <value>value1</value>
    <value>value2</value>
  </Attribute>
  <Attribute name="æøå">
    <value>æøå</value>
    <value>ÆØÅ</value>
  </Attribute>
</Entry>
```

ITDI XML Parsers

- ITDI supplies 4 (7.1) XML Parsers OOB
 - Simple XML Parser
 - Very simple to use – but limited in performance and functionality
 - StAX XML Parser
 - New and much improved "simple" parser based on StAX
 - XSL Parser
 - Ability to perform XSL transformations
 - SAX Parser
 - Can handle arbitrary complex XML documents

ITDI XML Utilities

- Class `com.ibm.di.util.XMLUtils` implements two handy methods :
 - `entry2XML(Entry e)`
 - `XML2Entry(String xml)`
- Class `com.ibm.di.eclipse.http.XML` from the eclipse framework implements :
 - `fromXML(org.w3c.dom.Document document or String xml)`
 - `toXML(java.lang.Object entryOrAttribute)`
- Class `UserFunction` (aka "system") implements a simple XSL Translator :
 - `xslTransform(Object xsl, Object xml)`

ITDI XML – The Entry class

- The Entry class
 - As of version 7.0 the Entry class is a subclass of `com.ibm.di.entry.DocImpl`
 - As such it implements DOM Document and Node interfaces
 - Has methods to work with Entry as a DOM object
 - Beware – not all DOM methods are implemented

XSL Transformation (XSLT)

- Language for transforming XML documents into other document formats including XML
- an XSLT stylesheet consists of a collection of template rules which define the transformations to be performed
- An XSLT stylesheet is itself an XML document
- ITDI implements a transformer in `system.xmlTransform(Object xsl, Object xml)`

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/
  Transform">
<xsl:output method="xml" encoding="UTF-8"/
  >
  <xsl:template match="node()|@*">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/
    >
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

The system.xsltTransform method

From the API documentation :

xsltTransform

```
public String xsltTransform(Object xsl,  
                           Object xml)
```

Calls the XSLTransformer to transform an XML document using a given style sheet. **"\n" needs to be present** in the XSL and XML string for xsltTransform to work correctly.

Parameters:

xsl - The XSL Style sheet (String, java.io.File, java.io.Reader)

xml - The XML document (String, java.io.File, java.io.Reader)

Returns:

The transformed document

A simple Identity transform

```
myXML = '<?xml version="1.0" encoding="UTF-8"?
  ><level1>some text<level2>E0Åæøå</level2></
  level1>' + "\n";
myXSL = '<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/
  Transform"><xsl:output method="xml"
  encoding="UTF-8"/><xsl:template match="node() |
  @*"><xsl:copy><xsl:apply-templates select="@* |
  node()" /></xsl:copy></xsl:template></
  xsl:stylesheet>' + "\n";

xmlOut = system.xmlTransform(myXSL,myXML);

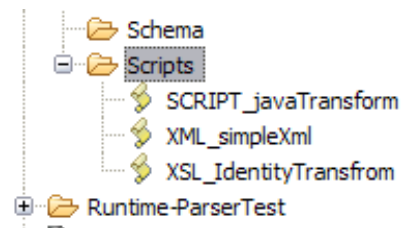
task.logmsg("xmlOut : \n" + xmlOut);
```

INFO - xmlOut :

```
<?xml version="1.0" encoding="UTF-8"?><level1>some
  text<level2>E0Åæøå</level2></level1>
```

Utilizing the ITDI script library

- XSL stylesheets are normally static string objects
- Multiline strings "clumsy" in JavaScript
- JavaScript does not provide "here documents" like e.g. Perl or PHP
- Reuse across assemblylines not easy
- If stored in the script library it can be retrieved using `system.getScriptText()` and easily reused
- Good practise is to use a naming scheme :



Utilizing the ITDI script library

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://
  www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" encoding="UTF-8"/>

  <xsl:template match="node()|@*">

    <xsl:copy>

      <xsl:apply-templates select="@*|node()"/>

    </xsl:copy>

  </xsl:template>

</xsl:stylesheet>
```

```
myXML = system.getScriptText("XML_simpleXml");
myXSL = system.getScriptText
  ("XSL_IdentityTransform");
xmlOut = system.xslTransform(myXSL,myXML);
task.logmsg("xmlOut : \n" + xmlOut);
```

```
<?xml version="1.0" encoding="UTF-8"?>

<level1>

some text

<level2>

ÆØÅæøå

</level2>

</level1>
```

```
INFO - xmlOut :
<?xml version="1.0" encoding="UTF-8"?><level1>
some text
<level2>
ÆØÅæøå
</level2>
</level1>
```

Managing the Output

- XSLT provides many options for managing the output
- Most important are :
 - method = "xml" | "html" | "text"
 - encoding = codepage
 - omit-xml-declaration = "yes" | "no"
 - cdata-section-elements = list of elements
 - indent = "yes" | "no"
- Examples :
 - `<xsl:output method="xml" indent="yes" encoding="utf-8" cdata-section-elements="filter" omit-xml-declaration="yes"/>`
 - `<xsl:output method="html" version="1.0" encoding="iso-8859-1" indent="yes"/>`

Managing the Output (2)

- ITDI does not honour output directives in all cases :
 - Using the XSL Parser the output directive is disregarded
 - The system.xsltTransform() disregards the indent output directive
 - use xalan output directive instead :
 - `<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xalan="http://xml.apache.org/xslt" version="1.0">`
`<xsl:output xalan:indent-amount="3"/>`
- Space options :
 - The `<xsl:preserve-space>` element is used to define the elements for which white space should be preserved.
 - The `<xsl:strip-space>` element is used to define the elements for which white space should be removed.

Beware of the Codepage

- Many transformations depends on the underlying codepage of the JRE
- JRE on Windows is NOT UTF-8.....
- Hence specify "UTF-8" on string conversions where applicable :
 - Java output stream : `os.toString("UTF-8")`
- Remember to specify the intended codepage on the XSL output directive
 - `<xsl:output encoding="utf-8">`
- Not all builtin utilities are codepage safe (yet) :
 - `XMLUtils.entry2XML()` (solved in next FP)

Transformation using Java method

- `system.xmlTransform()` does not allow other parameters passed
- To utilize the full potential it is necessary to use the java transformer classes and methods – e.g. `javax.xml.transformer`
- Requires definition of input and output streams and conversions to relevant `javax streamSource/ streamResult`
- Should manage codepage translation correctly

Transformation using Java methods (2)

```
function xslTransform( xsl, xml ) {
    os = new java.io.ByteArrayOutputStream();
    is = new java.io.ByteArrayInputStream(xsl.getBytes("UTF-8"));
    ss = new javax.xml.transform.stream.StreamSource(is);
    is1 = new java.io.ByteArrayInputStream(xml.getBytes("UTF-8"));
    ss1 = new javax.xml.transform.stream.StreamSource(is1);
    sr = new javax.xml.transform.stream.StreamResult(os);

    transformerfactory = javax.xml.transform.TransformerFactory.newInstance();
    transformer = transformerfactory.newTransformer(ss);

    // Set the output encoding
    outputEncoding = transformer.getOutputProperty("encoding")
    transformer.transform(ss1, sr);
    if (outputEncoding != null) {
        result = os.toString(outputEncoding);
    } else {
        result = os.toString();
    }
    return result;
}
```

Passing parameters to XSL

- To increase reuse of XSLT stylesheets it is necessary to be able to change behaviour based on runtime conditions e.g.
 - Only transforming a subset of the input
 - Changing formatting of output to depend on certain conditions
 - Changing output fields/buttons to include runtime variables such as servernames

Passing parameters to XSL(2)

```
function xslTransform( xsl, xml, params ) {
    os = new java.io.ByteArrayOutputStream();
    is = new java.io.ByteArrayInputStream(xsl.getBytes("UTF-8"));
    ss = new javax.xml.transform.stream.StreamSource(is);
    is1 = new java.io.ByteArrayInputStream(xml.getBytes("UTF-8"));
    ssl = new javax.xml.transform.stream.StreamSource(is1);
    sr = new javax.xml.transform.stream.StreamResult(os);

    transformerfactory = javax.xml.transform.TransformerFactory.newInstance();
    transformer = transformerfactory.newTransformer(ss);
    // Add paramaters to the transform
    for (i=0; i<params.length; i++) {
        transformer.setParameter((params[i])[0], (params[i])[1]);
    }
    // Set the output encoding
    outputEncoding = transformer.getOutputProperty("encoding");
    transformer.transform(ssl, sr);
    if (outputEncoding != null) {
        result = os.toString(outputEncoding);
    } else {
        result = os.toString();
    }
    return result;
}
```

Passing parameters to XSL(3)

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://
  www.w3.org/1999/XSL/Transform">

<xsl:output method="text" encoding="utf-8" omit-xml-
  declaration="yes"/>

<xsl:param name="ID">2</xsl:param>

<xsl:template match="@*|node() ">

<xsl:apply-templates select="//entry[@id=$ID]"/>

</xsl:template>

<xsl:template match="//entry[@id=$ID]">

<xsl:copy-of select="."/>

</xsl:template>

</xsl:stylesheet>
```

```
<entries>

  <entry id="1">

    this is text one

  </entry>

  <entry id="2">

    this is text two

  </entry>

  <entry id="3">

    this is text three

  </entry>

</entries>
```

Passing parameters to XSL(4)

```
myXML = system.getScriptText("XML_entriesXml");
myXSL = system.getScriptText("XSL_paramTransform");

myParam = [];
xmlOut = xslTransformWithParams(myXSL,myXML,myParam);

task.logmsg("Calling with no parameters - xmlOut :
\n" + xmlOut);

myParam = [{"ID","3"}];
xmlOut = xslTransformWithParams(myXSL,myXML,myParam);

task.logmsg("Calling with ID=3 parameter - xmlOut :
\n" + xmlOut);
```

```
INFO - Calling with no parameters - xmlOut :

this is text two

INFO - Calling with ID=3 parameter - xmlOut :

this is text three
```

Executing java from XSL

Overcoming the limitations of
XSLT

Executing java from XSL

- XSLT can be extended using *extension functions* and *extension elements*
- To utilize extension the following must be taken care of :
 - Binding the extension to the stylesheet
 - Mapping the five XSLT types (number, boolean, string, node-set, and result tree fragment) to Java types and vice versa

Binding an extension function

- Declaring a namespace for extension function :

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version = "1.0"
  xmlns:string="xalan://java.lang.String"/>
```

- Now the XSLT transformer can call java String methods using this syntax :

```
<xsl:variable name="myText" select="string:new(.)" />
<xsl:value-of select="string:toUpperCase($myText)"/>
```

Uppercasing using XSLT/Java

```
<html lang="en-us" xml:lang="en-us">
<body>
<h1>This is a heading</h1>
And here goes the text
</body>
</html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version = "1.0"
  xmlns:string="xalan://java.lang.String">

  <xsl:output method="html" indent="yes"/>

  <xsl:template match="/">
  <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="*">
    <xsl:copy>
      <xsl:apply-templates select="*|comment()|@*|text()"/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="@*">
    <xsl:attribute name="{name(.)}">
      <xsl:value-of select="normalize-space(.)"/>
    </xsl:attribute>
  </xsl:template>

  <xsl:template match="comment() ">
    <xsl:comment>
      <xsl:value-of select="normalize-space(.)"/>
    </xsl:comment>
  </xsl:template>

  <xsl:template match="text() ">
    <xsl:if test="string-length(normalize-space()) > 0">
      <xsl:variable name="myText" select="string:new(.)" />
      <xsl:value-of select="string:toUpperCase($myText)"/>
    </xsl:if>
  </xsl:template>

</xsl:stylesheet>
```

Uppercasing using XSLT/Java

The script and output

```
myXML = system.getScriptText("XHTML_simplePage");
myXSL = system.getScriptText
    ("XSL_XSLTWithJavaString");

myParam = [];
xmlOut = xslTransformWithParams(myXSL,myXML,myParam);

task.logmsg("All text into uppercase - xmlOut : \n" +
    xmlOut);
```

```
INFO - All text into uppercase - xmlOut :
<html lang="en-us" xml:lang="en-us">
<body>
<h1>THIS IS A HEADING</h1>
AND HERE GOES THE TEXT
</body>
</html>
```

Calling a ITDI library javascript function

- The ITDI resources are not directly available from XSLT
- But most ITDI resources are available utilizing the ITDI Java api
- It is possible to execute a ITDI script from XSLT

Calling the ITDI javascript engine from java

- Calling an ITDI script from java requires
 - The ITDI server object
 - `com.ibm.di.function.UserFunctions.getServer()`
 - Setting up the script Engine using the constructor
 - `new com.ibm.di.script.ScriptEngine(String scriptlang, RSInterface server)`
 - Loading the script from the library
 - `com.ibm.di.script.ScriptEngine.loadScript(RSInterface parent, String contextName, String name, boolean forceInclude)`
 - Calling the script function
 - `com.ibm.di.script.ScriptEngine.call(String function, Object[] param)`

Calling the ITDI javascript engine from java within XSLT

- The bindings :

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version = "1.0"
  xmlns:string="xalan://java.lang.String"
  xmlns:se="xalan://com.ibm.di.script.ScriptEngine"
  xmlns:itdi="xalan://com.ibm.di.function.UserFunctions" >
```

- Performing the call :

```
<xsl:template match="text()">
  <xsl:variable name="myServer" select="itdi:getServer()" />
  <xsl:variable name="mySE" select="se:new('javascript',$myServer)" />
  <xsl:variable name="temp0" select="se:loadScript($mySE,$myServer,'myContext',
    'myScriptFunction ','true')" />
  <xsl:if test="string-length(normalize-space(.)) > 0">
    <xsl:variable name="myParam" select="string:split(normalize-space(.),'$')" />
    <xsl:value-of select="se:call($mySE,'myScriptFunction',$myParam)"/>
  </xsl:if>
</xsl:template>
```

Example : Transform a HTML file

- Sometimes it would be nice to be able to transform a set of HTML files from/to your local language using e.g. Google Translate
- This example shows how to translate a single HTML file – it is trivial ITDI functionality to extend this to many

Example : Translate a HTML file

The components...

- This showcase involves :
 - A piece of HTML (stored as a script)
 - A script that calls Google Translate
 - A stylesheet that calls the translate script through XSLT java extensions
 - A script that performs the Steps
- Full source is included in the configuration downloadable with this presentation (ITDI 7.0 required)

Example : Translate a HTML file The Stylesheet...

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version = "1.0"
xmlns:string="xalan://java.lang.String"
xmlns:se="xalan://com.ibm.di.script.ScriptEngine"
xmlns:itdi="xalan://com.ibm.di.function.UserFunctions" >
... parts omitted
<xsl:template match="text()">
<xsl:variable name="myServer" select="itdi:getServer()" />
<xsl:variable name="mySE" select="se:new('javascript',$myServer)" />
<xsl:variable name="temp0" select="se:loadScript($mySE,
    $myServer,'myContext','SCRIPT_XSLTWithJavaGoogleTranslate','true')"/>
<xsl:if test="string-length(normalize-space(.)) > 0">
<xsl:comment><xsl:text> </xsl:text><xsl:value-of select="normalize-space(.)"/><xsl:text> </xsl:text></
    xsl:comment>
<xsl:text>
</xsl:text>
<xsl:variable name="myParam" select="string:split(normalize-space(.),'$')"/>
<xsl:value-of select="se:call($mySE,'googleTranslate',$myParam)"/>
<xsl:text>
</xsl:text>
</xsl:if>
</xsl:template>
</xsl:stylesheet>
```

Example : Translate a HTML file

The output...

```
myXML = system.getScriptText("XHTML_simplePage");
myXSL = system.getScriptText
    ("XSL_XSLTWithJavaGoogleTranslate");

myParam = [];
xmlOut = xslTransformWithParams(myXSL,myXML,myParam);

task.logmsg("All text translated into Danish -
    xmlOut : \n" + xmlOut);
```

```
INFO - All text translated into Danish - xmlOut :
<html lang="en-us" xml:lang="en-us">
<body>
<h1>
<!-- This is a heading -->
Dette er en overskrift
</h1>
<!-- And here goes the text -->
Og her g&aring;r teksten
</body>
</html>
```

DTD Validation

How to avoid validation without
retrieving external DTDs

ITDI and DTD validation

The problem

- Many XML documents include references to external DTD resources – e.g. XHTML files :

```
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

- Transforming such a document using `system.xmlTransform()` will go to the DTD url to get the DTD for validation
- W3 org was overloaded with calls due to this and hence responds with "HTTP 503"
- This will cause ITDI to abort execution.

ITDI and DTD validation

The Solution

- Use Java code instead of `xslTransform`
 - It is possible to specify that validation is not enabled or point the resolver to local resources
 - This is complex to handle and needs complex java code
- Use local validation
 - By setting up a catalog, property files and storing the DTDs locally ITDI will use these instead of going to external resources
 - Still need a custom transformer

Avoid DTD validation in java code

- Requires setting up an XMLReader
- XML reader requires SAX input source
- Both the XML and XSL might involve validation - so both input sources must be associated with the XMLReader

Avoid DTD validation in java code

```
function xslTransform( xsl, xml ) {  
  
    //Set up SAX inputSource for xsl stylesheet  
    ssr = new java.io.StringReader(xsl);  
    inps = new org.xml.sax.InputSource(ssr);  
  
    //Set up SAX inputSource for xml data  
    ssr1 = new java.io.StringReader(xml);  
    inps1 = new org.xml.sax.InputSource(ssr1);  
  
    //Set up XMLReader and turn off validation  
    xr = org.xml.sax.helpers.XMLReaderFactory.createXMLReader();  
    xr.setFeature("http://apache.org/xml/features/nonvalidating/load-external-dtd",false);  
  
    //Create SAXSource for xsl and associate it with the XMLReader  
    saxs = new javax.xml.transform.sax.SAXSource(xr,inps);  
    saxs.setXMLReader(xr);  
  
    //Create SAXSource for xml and associate it with the XMLReader  
    saxs1 = new javax.xml.transform.sax.SAXSource(xr,inps1);  
    saxs1.setXMLReader(xr);  
  
    //Create OutputStream for the transformation  
    os = new java.io.ByteArrayOutputStream();  
    sr = new javax.xml.transform.stream.StreamResult(os);  
  
    //Setup the transformer and transform...  
    transformerfactory = javax.xml.transform.sax.SAXTransformerFactory.newInstance();  
    transformer = transformerfactory.newTransformer(saxs);  
    outputEncoding = transformer.getOutputProperty("encoding")  
    transformer.transform(saxs1, sr);  
    if (outputEncoding != null) {  
        result = os.toString(outputEncoding);  
    } else {  
        result = os.toString();  
    }  
    return result;  
}
```

Perform local validation using a catalog

- Following files must be available to ITDI
 - CatalogManager.properties
 - Describes the catalogs and resolver properties
 - Catalogs for the different DTDs
 - The DTD Files
- Default location is `<ITDI_HOME>/classes`

Perform local validation using a catalog - examples

CatalogManager.properties

```
catalogs=xhtml.cat
relative-catalogs=false
static-catalog=yes
catalog-class-
    name=org.apache.xml.resolver.Resolver
verbosity=9
```

The catalog file (xhtml.cat)

```
<?xml version='1.0'?>
<catalog
  xmlns="urn:oasis:names:tc:entity:xmlns:xml:
  catalog">

  <public publicId="-//W3C//DTD XHTML 1.0
  Transitional//EN"
    uri="xhtml1-transitional.dtd"/>
  <public publicId="-//W3C//DTD XHTML 1.0
  Strict//EN"
    uri="xhtml1-strict.dtd"/>
  <public publicId="-//W3C//DTD XHTML 1.0
  Frameset//EN"
    uri="xhtml1-frameset.dtd"/>

</catalog>
```

Adding the Catalog to the transformer

- The following code will enable the catalog resolver :

```
....  
//Set up XMLReader and turn off validation  
    xr = org.xml.sax.helpers.XMLReaderFactory.createXMLReader();  
  
//Set up Catalog resolver - catalog.properties must be in <TDI_HOME>/classes  
//If validation is swithced off this will NOT be used...  
    cr = new org.apache.xml.resolver.tools.CatalogResolver();  
    xr.setEntityResolver(cr);  
....
```

The XML Input with DTD reference

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/
  DTD/xhtml1-transitional.dtd">
<html lang="en-us" xml:lang="en-us">
<body>
<h1>This is a heading</h1>
And here goes the text
</body>
</html>
```

Perform a transformation with external DTD checking enabled

```
myXML = system.getScriptText
("XHTML_simplePageWithValidation");
myXSL = system.getScriptText
("XSL_XSLTWithJavaGoogleTranslate");

myParam = [];
xmlOut = xslTransformWithParams(myXSL,myXML,myParam);

task.logmsg("All text translated into Danish -
xmlOut : \n" + xmlOut);
```

```
INFO - All text translated into Danish - xmlOut :
(no output)
```

From the console :

```
(Location of error unknown)java.io.IOException:
Server returned HTTP response code: 503 for URL:
http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd
```

Perform the a transformation with local DTD checking

```
myXML = system.getScriptText
    ("XHTML_simplePageWithValidation");
myXSL = system.getScriptText
    ("XSL_XSLTWithJavaGoogleTranslate");

xmlOut = xslTransformWithValidation(myXSL,myXML);

task.logmsg("All text translated into Danish -
    xmlOut : \n" + xmlOut);
```

```
All text translated into Danish - xmlOut :
<html lang="en-us" xml:lang="en-us">
<body>
<h1>
<!-- This is a heading -->
Dette er en overskrift
</h1>
<!-- And here goes the text -->
Og her g&aring;r teksten
</body>
</html>
```

Questions ?