XML Transformation Service – Documentation

1. General

Service used to transform XML based data using previously defined transformers.

This document is written for content developers that would like to work with XML based data and have basic knowledge in XSL.

Transformation Service is based on the sap XML toolkit that implements JAXP (see http http://java.sun.com/xml/jaxp/index.jsp).

The service provide some built-in transformers see detailed description below.

Transformation service supports all JAXP XML source and result formats. It also adds additional source, HTTPStreamSource, based on the content fetching service that provide caching abilities, proxy settings and access to properties defined in the PCD.

Transformation service interface: ITransformerService

Transformation service key: ITransformerService.KEY

2. Transforming XMLs

Transforming XMLs is the main feature of the Transformation service. Transformation can be done on previously registered transformers only.

Transformation is done using two types of provided transformers: XSL and SAX. Transformation service enables you to define your transformers once and then use them in combination with already defined transformers without the need to re-define them before transforming. Combination of several transformers to produce one output is called **Pipe Transformation**. Pipe transformation enables you to develop lite weight transformers and connect them together to one powerful transformer. Registering transformers also enables other content developers to work with the provided transformers using XML iViews UI directly from the portal.

Each transformer has the following attributes gathered by interface called *ITransformerInformation*:

- **Name:** The name of the transformer. Doesn't have to be unique per server, only per application component. String object.
- **Component name:** The name of the portal component or portal service that registered the transformer. String object.
- **Version:** The version of the transformer, each transformer might be upgraded; using the precise version will avoid upgrading errors. Float object.
- **Transformer type:** registration type of the transformer. *TransformerType* object. Can be one of the following:
  - o **Built-in:** Transformer was provided by the transformation service.
  - o **Persistent:** Transformer was supplied by transformation provider (see 3).
  - o **Temporary:** Transformer was temporary supplied from regular portal application.
- **From scheme:** XML Input format - from which scheme the transformer can get XML data. String object.
- **To Scheme:** XML output format – in which format will be the transformers output. String object.
- **Description:** General description of the transformer String object.

2.1. Activation steps:

2.1.1. **Building transformers list:** Getting related transformers information according to the attributes listed is done by inserting the requested attributes of transformer to the method *getTransformersInformation.* You should call this method once per transformer that you would like to insert. The method return more than one transformer information if more than one transformer is matching the attributes inserted. An alternative way of

building the list is to insert the transformers key to the transformers list. The transformer key can be retrieved from *ITransformerInformation* that was previously got. This option is not recommended since transformers can be removed by providers.

2.1.2. **Defining parameters:** Create array with the same size like the transformers list. Each item in the array must be a map of parameters related to transformer the transformer in the same index.

2.1.3. **Build the XML source:** Generate the XML that will be transformed. See XML Source.

2.1.4. **Build XML result:** Generate the object that will receive the result of the transformation. See **Error! Reference source not found.**XML Result.

2.1.5. **Activate transformation:** call method *transform* of the service.

2.2. XML Source

2.2.1. **General:** XML source must be a class derived from *javax.xml.transform.Source*. XML input can be loaded from file, stream, DOM or other customized class. Source class provided by the JDK are:

- *javax.xml.transform.stream,StreamSource*
- *javax.xml.transform.sax,SAXSource*
- *javax.xml.transform.dom,DOMSource*

See JDK Documentation for more details about these classes.

2.2.2. **HTTPStreamSource class:** is a class used for working with URL based XML sources that uses the portal cache, proxy settings and PCD attributes to load XMLs from the web. It is highly recommended to work with this class when loading XMLs from the web. For more details see class definition in the Javadocs and examples.

2.3. XML Result

2.3.1. **General:** XML result must be a class derived from *javax.xml.transform.result.* The class supplied will hold the result of the transformation in the format specified by the last transformer in the transformation pipe. Result classes provided by the JDK are:

- *javax.xml.transform.stream,StreamResult*
- *javax.xml.transform.sax,SAXResult*
- *javax.xml.transform.dom,DOMResult*

See JDK Documentation for more details about these classes.

2.3.2. **IPortalComponentResponse as transformation result:** in-order for the results to be written directly to the response of the portal component you should create a new StreamResult object with the response writer. See example:

…

```
                    // Setting result stream
StreamResult strmResult = new StreamResult(response.getWriter());
                    // Transforming
tService.transform(source, trns, paramsArray, context, null, strmResult);
```

2.3.3. **HTMLB result:** When working with the XHTMLB transformer the result of the transformation is a set of HTMLB objects. Default behavior is to create this set of objects in the page context, however it is possible to specify that

the result should be rendered to the response by parameters (see built-in transformers).

2.3.4. **String result:** In-order for the result to be written to a string you should create a StreamResult object with a ByteArrayOutputStream as its constructor input field. After transformation is done get the string from the ByteArrayOutputStream object you created. See example:

```
OutputStream outStrm = new ByteArrayOutputStream();

StreamResult strm = new StreamResult(outStrm); tService.transform(source, trns,
params, context, null, strm);

.

.

tService.transform(source, trns, params, context, null, strm);

.

.

String result = outStrm.toString();
```

3.Providing Transformers

3.1. General

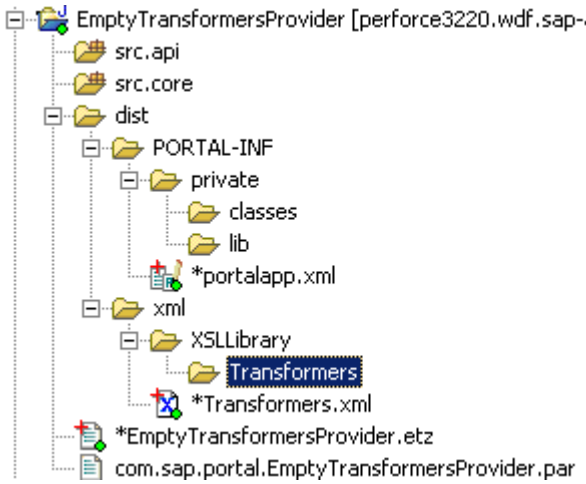
Transformers provider is a PAR file that holds transformers and is responsible for registering and removing the transformers from the list of transformers in the service. All the transformers will be registered in the service as persistent ones. The provider can hold as many transformers as needed. The provider can hold both XSL and SAX transformers.

Provider PARs default implementation is the class: *com.sap.portal.httpconnectivity.transformationservice. TransformersProvider.* All derivations for customizations are done from this class.

3.2. Simple providers

3.2.1. **General**: a provider that doesn't have to implement any Java code. Simple providers can include XSL transformers only, cannot provide language properties (ResourceBundle classes) and cannot customize the default behavior.

3.2.2. **PAR Structure**: The folder view should be as shown in the picture below:



The folder named “Transformers” is the one holding all the XSL files. Note that there are no Java files in the project. Implementing different folders tree needs customization of code (see below).

3.2.3. **Portalapp.xml file**: The portalapp.xml is responsible for registering the provider in the PRT registry mechanism, declaring the provider service configuration and general application configurations.

- **Portal registry** definition: defines this PAR as a provider in the registry entry. The *path* attribute must be: `runtime/transformers/com.sap.portal.EmptyTransformersProvider`, *`name`* `attribute must be` *`TransformersProvider`* `and` *`type`* `attribute must be` *`service.`* `Example:`

```
<registry>
  <entry path="runtime/transformers/com.sap.portal.EmptyTransformersProvider"
      name="TransformersProvider"
        type="service"/>
</registry>
```

- **Service configuration**: Declare the provider as a service. The name of the service must be *TransformersProvider*, the class name must be: *`com.sap.portal.httpconnectivity.transformationservice.TransformersProvider`*`. Note that the class name will be changed if the provider is not a simple one. Example:`

```
<services>
  <service name="TransformersProvider">
      <service-config>
      <property name="className"
value="com.sap.portal.httpconnectivity.transformationservice.TransformersProvider"/>
        <property name="classNameFactory" value=""/>
        <property name="classNameManager" value=""/>
      <property name="SecurityZone" value="com.sap.portal/no_safety" />
      </service-config>
    </service>
</services>
```

- **Application configuration**: set the *startup* property to true. This is done so that the PRT will register the provider when deploying and not in the first time that the service is called. Example:

```
<application-config>
    <property name="ServicesReference"
value="com.sap.portal.htmlb,com.sap.portal.transformationservice"/>
    <property name="releasable" value="false"/>
    <property name="startup" value="true"/>
</application-config>
```

3.2.4. **Transformers.xml file**: file holding the list of transformers provided. The file name should be “Transformers.xml” and should be located under <par folder>/dist/xml/transformers.xml. It is divided to two parts: XSL and SAX definitions. Each transformer has name, description, from/to scheme, source name and version (See ITransformerInformation above for more details about the properties). Source name property is the XSL file in case it is XSL transformer or a class name in case of SAX transformer.

Format example:

```
<?xml version="1.0" encoding="utf-8"?>
<transformation-resources>
  <transformers type="XSL"><!-- Holder of XSL transformers -->
    <transformer>     <!-- Represet single transforemr -->
      <property name="Name" value="MY_RSS_TO_XHTMLB"/> <!-- Key of the transformer-->
```

```
      <property name="Description" value="Transform RSS files to XHTMLB"/>
      <property name="FromURI" value="RSS"/>       <!-- Source scheme URI -->
      <property name="ToURI"     value="XHTMLB"/>    <!-- result scheme URI -->
      <property name="SourceName" value="RSS_TO_XHTMLB.xsl"/><!-- XSL file name -->
      <property name="Version" value="1.0"/>
    </transformer>
  </transformers>
  <transformers type="SAX"><!-- Holder of the SAX transformers -->
    <transformer>
      <property name="Name" value="MY_UNIQUE_ID_ADDER"/>
      <property name="Description"  value="Add unique id attribute to each node in the XML"/>

      <property name="FromURI" value="XML"/>
      <property name="ToURI" value="RSS"/>
      <property name="SourceName"
  value="com.sapportals.portal.httpconnectivity.saxtransformerprovider.MyUIDAdderHandler"/>
<!-- Full class name -->
      <property name="Version"     value="1.0"/>
    </transformer>
  </transformers>
</transformation-resources>
```

3.3. SAX providers

3.3.1. **General:** Providers that register a SAX transformer in the transformation service.

3.3.2. **Implementation**: Content developers must create their own class derived from *TransformersProvider* class and overwrite the method *getSAXHandler(ITransformerInformation info)* that returns new instance of SAX handler derived from *EPSAXDefaultHandler*. A simple implementation of the method will be to take the class name from the *ITransformerInformation* and instantiate.

3.3.3. **Portalapp.xml file**: The only change from simple providers will be the class name of the service.

3.4. Resource bundle holders

3.4.1. **General**: Provider that has some strings in the transformers that will need translation according to the language. The transformation service inserts a resource bundle object with the right language to each transformer as a parameter before transforming. The parameter name of the transformer is *"ResourceBundle".* When no resource bundle is defined the default transformation service one will be inserted.

3.4.2. **Implementation**: Content developers must create their own class derived from *TransformersProvider* class and overwrite the method *getResourceBundle(Locale locale)*. This method will be called in run-time before the transformation process is started. The concrete result class must be *PropertyResourceBundle*. The overwritten method must be in the core part of the PAR. The creation of the localization files is the same as in other PARs.

3.4.3. **Portalapp.xml file:** The only changes from simple providers will be the class name of the service and the property *ResourceBundleName* in the *service-config* part.

3.5. Customization options

All the customization options requires that the provider developer will implement a class that extends the *TransformersProvider* class and change the class name property in the portalapp.xml.

3.5.1. **Different folder tree**: In-order to place the Transformers.xml in a different location overwrite the method: *getTransformersResourcePath()*. This method returns a full path to a file holding the list of transformers. In-order to change the location of the XSL files overwrite the method: *getXSLTransformerPath(String tXSLPath).* This method receives the *SourceName* property from the Transformers.xml file and returns the full path to the XSL file.

3.5.2. **Add transformers from code**: Transformers providers can add transformers without the need to define them in the Transformers.xml file. To do this overwrite the *init(IServiceContext serviceContext)* method and implement in one of the following ways:

1. Call the default implementation, *super(serviceContext)*. Create the new transformer information using *createTransformerInsformation* method of the transformation service. Set the transformer by calling the appropriate *SetTransformer* method in the service.
2. Load transformers information from by calling *loadTransformers* method and hold the received list. Create your additional transformer using *createTransformerInformation* of the service and add the result ITransformerInformation to the list of transformers. Call method s*etTransformers* with the list of transformers information.

This customization is useful in cases that there is additional information that you need to add to each transformer or when your provider is service other purposes as well.

3.5.3. **Add packages to the SAX class names**: Overwrite method *String getSAXClassName(String tClassName)* that receive the value of the SourcePath property in the transformer definition in the transformers.xml file. This method must return a full class name including the package name.

3.5.4. **Stand alone provider**: Defined as a provider which is not derived from *com.sap.portal.httpconnectivity.transformationservice.TransformersProvider.* Developer of such provider must supply the following functionalities:

1. Provider must be a portal service.
2. Registration of the provider when deployed in PortalRegistry as described above.
3. Registration of the transformers when initialized. Transformers must be added using the *ITransformersService* methods.
4. Un-registration of the transformers when provider is removed or updated.
5. Un-registration from the PortalRegistry.

3.6. Creating provider step by step

Listed here are the steps needed to create a transformers provider PAR:

- Create a simple empty provider par as defined above or take the empty sample attached to this document.
- Edit potalapp.xml as described above.
- Edit the transformers.xml file.
- Create the XSL and SAX transformers and add them to the PAR.
- Implement SAX loader (if needed),
- Implement ResourceBundle provider and create localization files (if needed).
- Implement customize code (if needed).
- Deploy par.
- Your transformers should be available in the XML iView wizard and editor.

3.7. Declaring transformer parameters

3.7.1. **General**: When adding transformers the provider can define parameters that will control the behavior of the transformer. These parameters will be exposed in the XML iView editor. The rules that should be applied when providing parameters in transformer are listed below.

3.7.2. **Parameters types:** Provider can decide that some of its parameters shouldn't be exposed to the editor in these case the transformer shouldn't put the parameters in the list of parameters.

3.7.3. **SAX transformers parameters**: Each SAX handler must implement *ITransformerProperties* interface that declare one method: *getInputProperties()*. These method returns a map with the editable parameters of the SAX handler. The map keys are the names of the parameters. The values of the map can be of two types:

- String objects represent the default value.
- List of objects represent the valid options of the parameter. The first value in the list is the default one.

3.7.4. **XSL transformers parameters**: the parameters are declared in the regular XSL way using *xsl:param* elements. Controlling the type of the parameter is done using two additional attributes:

- sap:param-type: control whether this parameter should be editable. Possible values: *hidden* – non-editable parameter; *visible* - editable parameter. Default value is editable. Example: *<xsl:param name="ResourceBundle" sap:param-type="hidden"/>*
- sap:param-options: declares a list of possible values for the parameter. Syntax: sap:param-options="<default value>;<second value>;…". Example: *<xsl:param name="ShowDatesMode" sap:param-options="MAIN_ONLY;ALL;ITEMS_ONLY">ITEMS_ONLY</xsl:param>*

Note: When using XSL parameters you must declare their namespace: *xmlns:sap="http://www.sap.com/2004/Transformers/1.0"*.

4. Built-in transformers

4.1. General

Transformation service provides built-in transformers that give default solution to the following formats:

1. **XHTMLB** – see specification below
2. **RSS** – see http://web.resource.org/rss/1.0/spec
3. **Busdoc** XMLs.

All built-in transformers component name is *ITransformerService.BUILT_IN_TRANSFORMERS_KEY* and the transformer type is *BUILT-IN*.

4.2. XHTMLB Transformer

4.2.1. **General description**: SAX transformer. Transforms between XHMLB (see specification below) and HTMLB DOM. The transformer has no result in the regular way: the result of the transformation is a set of HTMLB objects ready to be rendered or to be included to other HTMLB objects. The user can supply HTMLB container (see parameters below) that will be used as the basic container for all the created HTMLB objects. If no top container is supplied the transformer will create HTMLB form document and insert all the HTMLB objects to the documents.

Id will be created to all the components when no ID attribute is given the XHTMLB element.

This transformer must be the last one in the list of transformers in the pipe.

Full specification and examples of the XHTMLB can be found below.

4.2.2. **From scheme**: XHTMLB

4.2.3. **To scheme**: HTML

4.2.4. **Name**: XHTMLBSAXHandler

4.2.5. **Version**: 1.0

4.2.6. **Parameters** (all the parameters are in the format *ITransformerService.BUILT_IN_TRANSFORMERS_KEY.<parameter name>*):

4.2.6.1. Parameter name: **DebugMode**

- Description: Flag indicating that debug logs and tracers should be written to the end of the top container. The log will include: errors notifications, trace of every component that was created and general status notifications.
- Value type: Boolean
- Default value: false
- Remarks: When the flag is set to True the performance will be worsted.

4.2.6.2. Parameter name: **HTMLBTopContainer**

- Description: The top container that all the HTMLB objects that are created in the process of the XHTMLB document will be inserted to.
- Value type: class derived from: com.sapportals.htmlb.Container
- Default value: none.
- Remarks: When no value is inserted by this parameter the transformer will try to get the HTMLB document from the page context, if no such document exists it will create formed document as the top container. This new created document is available from the page context (method *IPageContext.getDocument()*).

4.2.6.3. Parameter name: **HTMLBDeclaredComponents**

- Description: Parameter that defines a Map object that will be a holder of all the components with a given ID. This object can be later used to get this objects and manipulate them.
- Value type: java.util.Map
- Default value: null
- Remarks: When no object is defined, declared components will be treated as any other components.

4.2.6.4. Parameter name: **HTMLBRenderAtEnd**

- Description: Flag that define whether to render the created document to the response at the end of the transformation.
- Value type: Boolean
- Default value: True

4.2.7. **Error handling**: all the errors and the traces are written only in debug mode and printed as plain HTML at the end of the top container. No exception is thrown when trying to use non-existing methods or parameters.

4.2.8. **Non-supported HTMLB components**:

- MenuBar

- MenuItem
- Chart
- FormLayout
- DragSource
- DropTarget
- EventValidationComponent
- EventValidationContainer
- JavaScriptFragment
- PopupTrigger
- RoadMap
- RoadMapItem

4.3. RSS Transformer

4.3.1. **General description**: Transforms RSS feeds to XHTMLB.

4.3.2. **From scheme**: RSS (uri : http://purl.org/rss/1.0/) with Dublin-Core (uri: http://purl.org/dc/elements/1.1/) definitions of items and in RDF format of RSS feeds (uri: http://www.w3.org/1999/02/22-rdf-syntax-ns#).

4.3.3. **To scheme**: XHTMLB (see documentation below).

4.3.4. **Name**: RSS_TO_XHTMLB

4.3.5. **Version**: 1.0

4.3.6. **Parameters**:

4.3.6.1. Parameter name: **LinksTarget**

- Description: Define the target frame of the links in the RSS.
- Value type: String
- Default value: "_RSSItemWindow"

4.3.6.2. Parameter name: **ShowDatesMode**

- Description: Define the type of dates that will be rendered.
- Value type: String.
- Possible values: MAIN_ONLY – only the feed update date will be shown; ALL – all the dates will be shown; ITEMS_ONLY – only the items dates will be shown.
- Default value: ITEMS_ONLY.

4.3.6.3. Parameter name: **ShowTime**

- Description: Defines whether to show the time of each item.
- Value type: xs:boolean ("true" or "false").
- Default value: true
- Remarks: The time showing is affected by ShowDatesMode parameter as well.

4.3.6.4. Parameter name: **GroupByDate**

- Description: Defines whether to create groups according to the items dates. When ShowDatesMode is set to MAIN_ONLY this parameter is obsolete.
- Value type: xs:boolean ("true" or "false").
- Default value: true

4.3.6.5. Parameter name: **DateFormat**

- Description: Defines the output format of the date. The syntax is defined in java.text.SimpleDateFormat class documentation is the JDK.
- Value type: String.
- Default value: “EEEE, MMM dd, yyyy”
- Remarks: The actual formatting is done using the object supplied in by xslDateFormatter parameter.

4.3.6.6. Parameter name: **TimeFormat**

- Description: Defines the output format of the time. The syntax is defined in the java.text.SimpleDateFormat class documentation is the JDK.
- Value type: String.
- Default value: “HH:mm:ss a”
- Remarks: The actual formatting is done using the object supplied in by xslDateFormatter parameter.

4.3.6.7. Parameter name: **xslDateFormatter**

- Description: A parameter holding the class that formats the input date string by the output date/time formats.
- Value type: class com.sap.portal.httpconnectivity.transformationservice.xslextensions.XSLDateFormatter
- Default value: an instance of the class.
- Remarks:

4.3.6.8. **Parameter name**: ScrollHeight

- Description: The height of the scroll area of the items list.
- Value type: integer
- Default value: 300

4.4. Busdoc Transformer

4.4.1. **General description**: Transforms Busdoc XMLs to XHTMLB. The transformer supports all the data set types (HRField, HRScript, HRFreeText and HRRow). It also supports navigation within the table view of the data. HRNP links and “Drag & Relate” feature are not supported.

4.4.2. **From scheme**: Busdoc.

4.4.3. **To scheme**: XHTMLB.

4.4.4. **Name**: BUSDOC_TO_HTMLB

4.4.5. **Version**: 1.0

4.4.6. **Parameters**:

4.4.6.1. Parameter name: **`RenderDataAsHtml`**

- Description: define whether to render table data as HTML or just as plain data.
- Value type: Boolean.
- Default value: true

4.4.6.2. Parameter name: **VisibleRowsCount**

- Description: define the number of visible rows in the table.

- Value type: Integer.
- Default value: 10

5. XHTMLB Specification

5.1. General

XHTMLB is XML based format for creating HTMLB documents using the XHTMLBTransformer. Creating XHTMLB documents requires knowledge in HTMLB.

The XHTMLB is transformed to HTMLB component using XHTMLBTransformer (see details above).

5.2. Regular controls Format

5.2.1. **HTMLB Components declaration:**

Each node in XHTMLB is equivalent to HTMLB component. Each attribute in XHTMLB element is equivalent to calling a set method with the name of the attribute as the name of the method and the attribute value as the input field.
The text inside the XML node is inserted to the component using the component method *setText* or *setValue*. If the current component is also HTMLB container, the method *addText* will be called. Only the last text node will be taken as the value.
Example XHTMLB node:

<Button id=”testButton” OnClick="MyEventValue">My button</Button>

Will create HTMLB button component with the text “My button” and with the value “MyEventValue” that will be sent when clicking the button.

Example of multiple text nodes:

```
<TextView Wrapping="true">
This text will be overwritten
<parameter name="Design" value="LABEL" class="enum.TextViewDesign"/>
This text will appear in the TextView component
</TextView>
```

5.2.2. **Parameters declaration:**

Declaring parameter is done for two reasons:

- Calling a method on the current component that has a static input field value.
- Adding key/value pair to the current component (when needed).

Calling a method that requests a static value from class (like enumerated values) is done by adding element named “parameter” with the attributes:

Name – the name of the method without the “set” or “add” prefix; class – the name of the class; package – the name of the package (default package is com.sapportals.htmlb); value – the name of the static member that will be inserted to the method.

Example for calling method with static field of a class as a value:

```
<TextView Wrapping="true">
<parameter name="Design" value="LABEL" class="enum.TextViewDesign"/>
<parameter name="Layout" value="BLOCK" class="enum.TextViewLayout"/>
Some text with label design and block layout
</TextView>
```

In this example HTMLB TextView component will be created. The method *setDesign* will be called with com.sapportals.htmlb.enumTextViewDesign.LABEL class as an input field. The method *setLayout* will be called with

com.sapportals.htmlb.enumTextViewLayout.BLOCK class as an input field.

Adding key/value pair is done on specific HTMLB components like ListBox, DropDownListBox, ToolbarDropDownListBox and BreadCrumb.

Example:

```
<DropdownListBox id="Dropdown1" width="120"          text="Dropdown">
  <parameter key="Item1" value="Value one"/>
  <parameter key="Item2" value="Value two"/>
</DropdownListBox>
```

5.2.3. **HTML elements**:

Regular HTML elements will be inserted to the current HTMLB container. There is no way to insert HTMLB components inside HTML elements, once HTML element was defined all its child will be HTML elements as well.

Example:

```
<Group>
  <ButtonRow>
    <Button id="tst7" width="100">Test1</Button>
    <Button id="tst8" width="100">Test2</Button>
  </ButtonRow>
  <span>
    <a href="http://www.cnn.com">HTML Link</a>
  </span>
</Group>
```

In this example the span and its children will be inserted to the HTMLB Group container.

5.2.4. **Components hierarchy**:

The hierarchy between the components and the containers is defined by the hierarchy in the XML document. The restrictions on the each component are according to the restrictions in HTMLB.

5.2.5. **Simple example**:

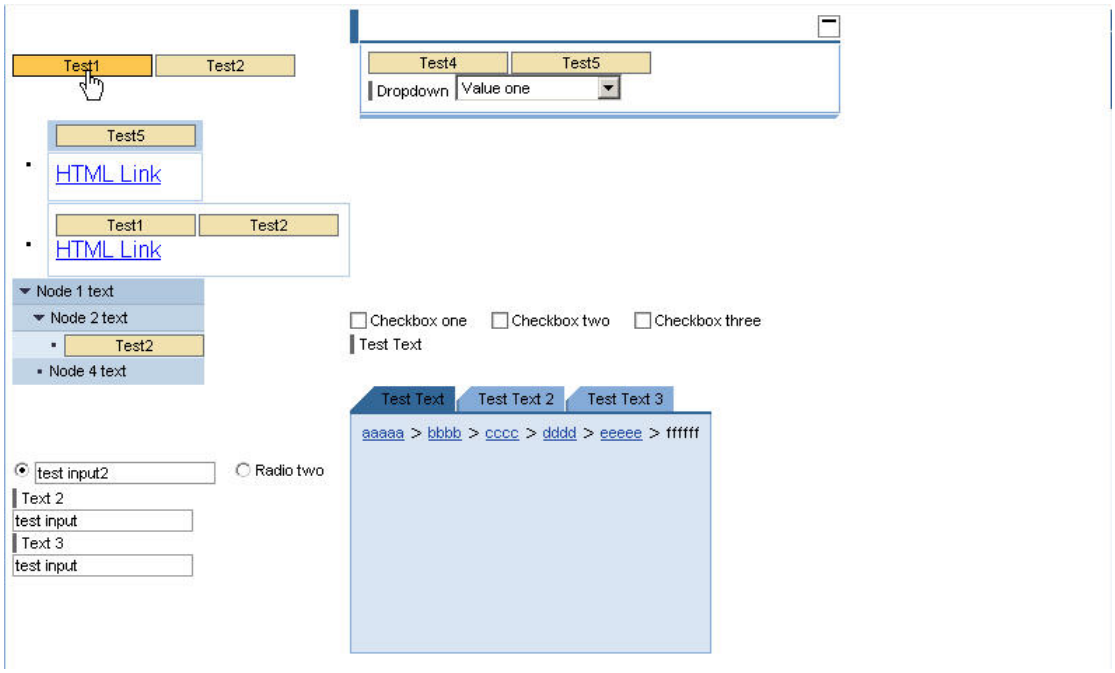
XHTMLB code:

```
<?xml version="1.0"?>
<GridLayout>
  <GridLayoutCell id="cell1" row="1" column="1" width="100" height="100">
    <ButtonRow>
      <Button id="tst1" width="100">Test1</Button>
      <Button id="tst2" width="100">Test2</Button>
    </ButtonRow>
  </GridLayoutCell>
  <GridLayoutCell id="cell2" row="1" column="2" width="100" height="100">
    <Tray>
      <ButtonRow>
        <Button id="tst3" width="100">Test4</Button>
        <Button id="tst4" width="100">Test5</Button>
      </ButtonRow>
      <DropdownListBox id="Dropdown1" width="120" text="Dropdown">
        <parameter key="Item1" value="Value one"/>
        <parameter key="Item2" value="Value two"/>
      </DropdownListBox>
    </Tray>
  </GridLayoutCell>
  <GridLayoutCell id="cell3" row="2" column="1" width="100" height="100">
      <ItemList>
        <Group tooltip="group one tooltip">
          <Button isHeaderComponent="true" id="tst6" width="100">Test5</Button>
          <a href="http://www.cnn.com">HTML Link</a>
```

```
        </Group>
        <Group>
          <ButtonRow>
            <Button id="tst7" width="100">Test1</Button>
            <Button id="tst8" width="100">Test2</Button>
          </ButtonRow>
          <a href="http://www.cnn.com">HTML Link</a>
        </Group>
      </ItemList>
  </GridLayoutCell>
  <GridLayoutCell id="cell3" row="3" column="1" width="100" height="100">
    <Tree id="Tree1">
      <TreeNode id="Node1" Text="Node 1 text">
        <TreeNode id="Node3" Text="Node 2 text">
          <TreeNode id="Node2" Text="Node 3 text">
            <Button id="tst9" width="100">Test2</Button>
          </TreeNode>
        </TreeNode>
        <TreeNode id="Node4" Text="Node 4 text"/>
      </TreeNode>
    </Tree>
  </GridLayoutCell>
  <GridLayoutCell id="cell4" row="3" column="2" Width="100%" height="100">
    <FlowLayout>
      <CheckboxGroup id="CheckboxGroup1" ColumnCount="6">
        <Checkbox id="Checkbox1" value="Checkbox one"/>
        <Checkbox id="Checkbox2" value="Checkbox two"/>
        <Checkbox id="Checkbox2" value="Checkbox three"/>
      </CheckboxGroup>
      <TextView Labeled="true" Text="Test Text"/>
    </FlowLayout>
  </GridLayoutCell>
  <GridLayoutCell id="cell5" row="4" column="1" width="100" height="100">
    <FlowLayout>
      <RadioButtonGroup id="RadioGroup1" key="TestRadio" ColumnCount="10">
        <RadioButton id="Radio1" key="TestRadio0" selected="true"><InputField sid="Input0"
value="test input2"/></RadioButton>
        <RadioButton id="Radio2" key="TestRadio1" text="Radio two"
Labled="true"></RadioButton>
      </RadioButtonGroup>
      <InputField id="Input2" value="test input" text="Text 2"/>
      <InputField id="Input3" value="test input" text="Text 3"/>
    </FlowLayout>
  </GridLayoutCell>
  <GridLayoutCell id="cell6" row="4" column="2" width="100" height="100">
    <TabStrip id="TabStrip1" width="200" bodyHeight="170">
      <TabStripItem index="1" title="First Tab">
        <TextView isHeader="true">Test Text</TextView>
        <BreadCrumb isBody="true" id="BreadCrumb1" OnClick="alert('aaa');">
          <parameter name="Size" value="SMALL" class="enum.BreadCrumbSize"/>
          <parameter name="Behavior" value="SINGLELINK"
class="enum.BreadCrumbBehavior"/>          <parameter key="aaaa" value="aaaaa"/>
          <parameter key="bbbb" value="bbbb"/>
          <parameter key="ccc" value="cccc"/>
          <parameter key="dddd" value="dddd"/>
          <parameter key="eeee" value="eeeee"/>
          <parameter key="ffff" value="ffffff"/>
        </BreadCrumb>
      </TabStripItem>
      <TabStripItem index="2" title="Second Tab">
```

```
        <TextView isHeader="true">Test Text 2</TextView>
        <ScrollContainer isBody="true" width="100" height="100">
          <Image src="http://www.ynet.co.il/PicServer/02202003/253858/poraz_a.jpg"
width="150" height="150">
            <ImageMap id="ImageMap1">
              <ImageArea Coordinates="0,0,150,50"><Link reference="http://www.cnn.com"
target="_blank"/></ImageArea>
              <ImageArea Coordinates="0,50,150,100"><Link reference="http://www.bbc.com"
target="_blank"/></ImageArea>
              <ImageArea Coordinates="0,100,150,150"><Link
reference="http://www.latimes.com" target="_blank"/></ImageArea>
            </ImageMap>
          </Image>
        </ScrollContainer>
      </TabStripItem>
      <TabStripItem index="3" title="Third Tab">
        <TextView isHeader="true">Test Text 3</TextView>
        <DateNavigator id="DateNavigator1" text="Date navigator"
  MonthsPerColumn="1" MonthsPerRow="1"
onDayClick="xxxxxxxxxxxxx"><DateNavigatorModel CnteredMonth="2" centeredYear="2004"
localeUnknown="true"/></DateNavigator>
      </TabStripItem>
    </TabStrip>
  </GridLayoutCell>
</GridLayout>
```

Appearance in the portal:



5.3. Creating XHTMLB tables

5.3.1. **General structure**: XHTMLB table are created by defining TableView element with JCOTableViewModel or DefaultTableViewModel below it. The models can have TableColumn elements and non-htmlb TableRow elements. TableRow elements can have TableCell elements which are non-htmlb elements as well. Structure example:

```
<TableView ...>
  <JCOTableViewModel ...>
    <TableColumn index="0" name="FirstRow" ...>
    </TableColumn>
    <TableColumn index="1" name="SecondRow" ...>
```

```
    </TableColumn>
    <TableRow ...>
      <TableCell>Row one col one</TableCell>
      <TableCell>Row one col two</TableCell>
    </TableRow>
    <TableRow>
      <TableCell>Row two col ones</TableCell>
      <TableCell>Row two col two</TableCell>
    </TableRow>
    <TableRow>
    </TableRow>
    ...
  </JCOTableViewModel>
</TableView>
```

5.3.2. **TableView element**: can have all the “set” methods (XHTMLB attributes and parameter elements) like a regular HTMLB element except the following:

- *setModel* – used in a different way than HTMLB. It is used to insert TableViewModel by parameter. When TableView has *model* attribute, the value must point to a parameter inserted to the XHTMLB transformer with *TableViewModel* derived object.
- *CellRenderer* – XHTMLB writer can define his own cell-render. The cell render object must be derived from com.sap.portal.transformationservice.xhtmlb.IXHTMLBCellRenderer. If no cell renderer is defined than the data will be treated as raw data. The definition can be done by adding a parameter element to the TableView element with the class definitions in the *class* and *package* attributes or with the name of the external property in the *value* attribute. Only one cell renderer can be defined for a table, the last one specified will be the one in use. Each cell that should be marked with the attribute *useRenderer=”true”.* Example 1:

```
<TableView id="TestTable" width="400">
      <parameter name="CellRenderer"
      package="com.sap.portal.httpconnectivity.transformationservice"
      class="xhtmlb.DefaultXHTMLBCellRenderer"/>
      …
      <TableRow selectRow="true">
        <TableCell>http://www.cnn.com</TableCell>
        <TableCell useRenderer="true">
            <![CDATA[<div style="font-size:x-small">

              <a href="http://www.google.com">google</a>

              </div>]]>

        </TableCell>

    </TableRow>

</TableView>
```

- Row specific methods must be written in the TableRow element.
- Cell specified methods must be written in the TableCell element. Cell methods are the ones getting row index, cell index and a value to set.

5.3.3. **JCOTableViewModer/DefaultTableViewModel elements**: no attributes and parameters can be given to the model.

5.3.4. **TableColumn element**: support all the “set” methods of the HTMLB TableColumn component. Each table column must have an index (starting from 0) and a name. TableColumn doesn’t have child elements.

5.3.5. **TableRow element**: An element that doesn’t have HTMLB correspondent component. Used to wrap the content of a table row. It can receive all the methods related to rows. Row methods: *selectRow, setOnRowSelection, setRowSelectable, setRowVAlignment.*

5.3.6. **TableCell element**: An element that doesn’t have HTMLB correspondent component. The element holding the actual data of the cell. Can have “set” methods attributes of the TableView component that refers to cells. These methods take row index, column index and value to set. List of methods: *setCellDisabled, setCellHAlignment, setCellInvalid, setCellType, setCellVAlignment, setColspanForCell, setRowspanForCell, setStyleForCell.*

5.3.7. **Table Toolbar**: the of toolbar element must be before the starting declaration of TableView. The connection is done by assigning ID too the Toolbar element and adding toolbar=”<toolbar element ID>” in the TableView element.

5.3.8. **Limitations**:

- Cell cannot have child elements, meaning that no HTMLB component can be a child of table cell.
- All the table columns must appear before the table rows in the XHTMLB defining the table.
- Sorting is not supported by default.

5.3.9. **Example**:

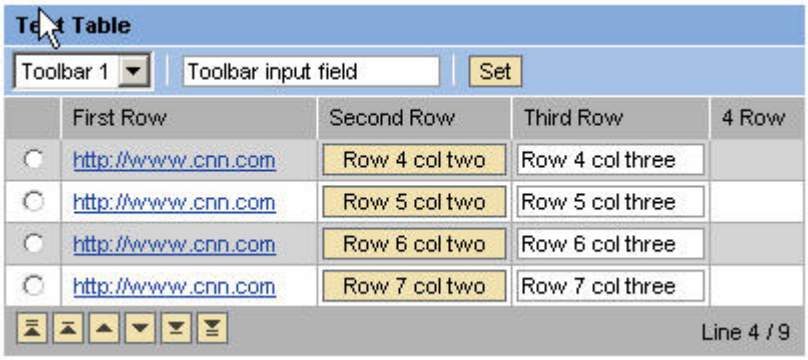
Example code:

```
<?xml version="1.0"?>
<FlowLayout>
  <Toolbar id="TestTableToolbar">
    <ToolbarDropDownListBox id="tbrLstBox">
      <parameter key="TestVal1" value="Toolbar 1"/>
      <parameter key="TestVal2" value="Toolbar 2"/>
    </ToolbarDropDownListBox>
    <ToolbarSeparator/>
    <ToolbarInputField id="tbrInpFld" value="Toolbar input field"/>
    <ToolbarSeparator/>
    <ToolbarButton id="tbrBttn" text="Set" value="SomeValue"/>
  </Toolbar>
    <TableView id="TestTable" width="400"
      Summary="Table summary" footerVisible="true"
      VisibleFirstRow="5" VisibleRowCount="4"
      OnNavigate="NavigateTestTable" HeaderText="Test Table"
      OnHeaderClick="DoClicked" toolbar="TestTableToolbar">
      <parameter name="CellRenderer"
package="com.sap.portal.httpconnectivity.transformationservice"
class="xhtmlb.XHTMLBCellRenderer"/>
      <parameter name="NavigationMode" value="BYLINE"
class="enum.TableNavigationMode"/>
      <parameter name="Design" value="ALTERNATING" class="enum.TableViewDesign"/>
      <parameter name="SelectionMode" value="SINGLESELECT"
class="enum.TableSelectionMode"/>
      <JCOTableViewModel>
        <TableColumn index="0" name="FirstRow" title="First Row"
```

```
          LinkClickTarget="_blank" LinkColumnKey="FirstRow" TooltipForColumnHeader="First
row tooltip">
          <parameter name="Type" value="LINK" class="enum.TableColumnType"/>
        </TableColumn>
        <TableColumn index="1" name="SecondRow" title="Second Row" CellsDragable="true">
          <parameter name="Type" value="BUTTON" class="enum.TableColumnType"/>
        </TableColumn>
        <TableColumn index="2" name="ThirdRow" title="Third Row">
          <parameter name="Type" value="INPUT" class="enum.TableColumnType"/>
          <parameter name="DropTargetDesign" value="BORDERED"
class="enum.DropTargetDesign"/>
        </TableColumn>
        <TableColumn index="3" name="4Row" title="4 Row" OnDrop="DropOnRow4"
Width="400"
          LinkClickTarget="_blank" LinkColumnKey="4Row">
          <parameter name="Type" value="TEXT" class="enum.TableColumnType"/>
          <parameter name="DropTargetDesign" value="UNDERLINED"
class="enum.DropTargetDesign"/>
        </TableColumn>
        <TableRow>
          <TableCell>Row one col one</TableCell>
          <TableCell>Row one col two</TableCell>
          <TableCell>Row one col three</TableCell>
          <TableCell>Row one col three</TableCell>
        </TableRow>
        <TableRow>
          <TableCell>Row two col ones</TableCell>
          <TableCell>Row two col two</TableCell>
          <TableCell colspanForCell="2">Row two col three</TableCell>
        </TableRow>
        <TableRow selectRow="true">
          <TableCell>http://www.cnn.com</TableCell>
          <TableCell>Row 3 col two</TableCell>
          <TableCell>Row 3 col three</TableCell>
          <TableCell useRenderer="true">
            <![CDATA[<div style="font-size:x-small">
              <a href="http://www.google.com">google</a>
            </div>]]>
          </TableCell>
        </TableRow>
        <TableRow>
          <TableCell>http://www.cnn.com</TableCell>
          <TableCell>Row 4 col two</TableCell>
          <TableCell>Row 4 col three</TableCell>
        </TableRow>
        <TableRow>
          <TableCell>http://www.cnn.com</TableCell>
          <TableCell>Row 5 col two</TableCell>
          <TableCell>Row 5 col three</TableCell>
        </TableRow>
        <TableRow>
          <TableCell>http://www.cnn.com</TableCell>
          <TableCell>Row 6 col two</TableCell>
          <TableCell>Row 6 col three</TableCell>
        </TableRow>
        <TableRow>
          <TableCell>http://www.cnn.com</TableCell>
          <TableCell>Row 7 col two</TableCell>
          <TableCell>Row 7 col three</TableCell>
        </TableRow>
```

```
        <TableRow>
          <TableCell>http://www.cnn.com</TableCell>
          <TableCell>Row 8 col two</TableCell>
          <TableCell>Row 8col three</TableCell>
        </TableRow>
        <TableRow>
          <TableCell>http://www.cnn.com</TableCell>
          <TableCell>Row 9 col two</TableCell>
          <TableCell>Row 9 col three</TableCell>
        </TableRow>
      </JCOTableViewModel>
    </TableView>
</FlowLayout>
```

Result view:



5.4. Limitations

5.4.1. **Not supported HTMLB components**:

- MenuBar
- MenuItem
- Chart
- FormLayout
- DragSource
- DropTarget
- EventValidationComponent
- EventValidationContainer
- JavaScriptFragment
- PopupTrigger
- RoadMap
- RoadMapItem

5.5. Exceptions

5.5.1. **General:** All the listed below are methods/parameters/components that behave different from the general description above. Note that the TableView and its related components are written differently altogether.

5.5.2. **Group**: In order to set one of the group child elements as it's header component add attribute *isHeaderComponent="true"* to the child element.

5.5.3. **TabStripItem:** TabStripItem child elements that should be the header component must have *isHeader="true"* attribute. Other child elements will be inserted to the body of the TabStripItem by the order of appearance.

5.5.4. **ItemList**: Each component in the list can declare its bullet image URL by adding *bulletURI="<bullet URL>"* attribute. Each component in the list can declare it's bullet style by adding *style="<style parameters"* attribute.

5.5.5. **GridLayoutCell**: All these elements must have *row* and *column* attribute with integer value starting from 1.

6.XSL Extensions

6.1. XSLDateFormatter

6.1.1. **Purpose:** format dates to a given output formats.

6.1.2. **Class name**: com.sap.portal.httpconnectivity.transformationservice.xslextensions.XSLDateFormatter.

6.1.3. **Methods**:

6.1.3.1. public static String formatDate(String date, String format)

Method that formats the date given in the *date* parameter by format given in the *format* parameter. Dates format are according to the specifications that can be found in JDK documentation about *java.text.SimpleDateFormat* class. Output format can be any format (that follows the . Input date format must be one from the following:

- "EEE, dd MMM yyyy kk:mm:ss z" – for example Wed, 03 Dec 2003 07:10:05 GMT
- "yyyy-MM-dd'T'kk:mm-ss:SS" – for example 2004-02-03T20:53-08:00
- "yyyy-MM-dd'T'kk:mm:ss" – for example 2004-01-12T01:37:54-400

6.1.4. **Usage Example**: This example is taken from the RSS_TO_HTMLB transformer and used to normalize the date formats coming from the input.

```
<?xml version="1.0"?>
<xsl:stylesheet
      version="1.0" ...
  xmlns:transDate="com.sap.portal.httpconnectivity.transformationservice.xslextensions.XSLDateFormatter"
    >
…
  <xsl:template match="*" mode="EPRSS:Date">
    <xsl:variable name="DateStr"><xsl:value-of select="."/></xsl:variable>
    <xsl:value-of name="formatted"
  select="transDate:formatDate($DateStr,$TimeFormat)"/>
  </xsl:template>
…
```

7. Appendixes

7.1. Javadocs – see release documentation.

7.2. Code examples

7.2.1. **Simple empty provider project**: use this example as a basis for your implementation.

7.2.2. **Simple SAX provider**

7.2.3. **XSL with resource bundle provider**

7.2.4. **Simple RSS transformation portal component**