Applies to:
SAP for Banking. For more information, visit the Web Services homepage.

Summary
The purpose of this document is to describe in detail on how to extract the product attributes in XML in a pre-defined schema provided by the customer. In most of the cases, SAP Banking Services platform is used as a robust back end system for core banking and hence there might be lot of front end non-SAP applications interacting with this platform using web services. One such scenario is to extract all the attributes of a given product in XML format on a pre-defined XML Schema.

Author: Joshy Thampi Pulikotil
Company: SAP Global Delivery Center, Bangalore
Created on: 20 November 2008

Author Bio
Joshy Thampi Pulikotil is a technical consultant working in the area of SAP Financial Services. He is currently working at SAP Global Delivery Center, Bangalore as a part of the financial services team. He has got more than 6 years of experience in ABAP development. Currently his focus areas are SAP Deposit Management and Enterprise SOA.
# Table of Contents

Purpose ...............................................................................................................................................................3
Abstract ...............................................................................................................................................................3
Description ..........................................................................................................................................................3
  Product and Product Attributes in Banking Services Platform .................................................................3
  Definition of XML Schema using Package iXML ..........................................................................................5
  Package DOM .................................................................................................................................................6
  Package Stream ..............................................................................................................................................6
  Package Renderer ..........................................................................................................................................7
  Product Attributes Extract as XML file ............................................................................................................7
  Creation of Enterprise Services from RFC function module .................................................................8
Results ..............................................................................................................................................................10
Related Content ................................................................................................................................................11
Copyright ...........................................................................................................................................................12
Purpose

The purpose of this document is to describe in detail on how to extract the product attributes in XML in a predefined schema provided by the customer. In most of the cases, SAP Banking Services platform is used as a robust back end system for core banking and hence there might be lot of front end non SAP applications interacting with this platform using web services. One such scenario is to extract all the attributes of a given product in XML format on a pre-defined XML Schema. The biggest challenge here is to align our extract to the Schema defined by the customer. We already have the transform functionality to convert ABAP to XML but it is quite tedious to transform ABAP output content to XML in a pre-defined XML Schema. Thanks to SAP’s XML Library and Package DOM which will help us to resolve this by simply following few steps as described in this document.

Abstract

This document illustrates the following topics in detail.

- Product and Product Attributes in Banking Services Platform
- Definition of XML Schema using Package iXML
- Package DOM
- Package Stream
- Package Renderer
- Product Attributes Extract as XML file
- Creation of Enterprises Services from RFC function module

Description

Product and Product Attributes in Banking Services Platform

In Financial Services, Products form the basis for all business operations. Each contract is based on a product to which particular characteristics and functions are ascribed in general business transactions. An attribute offers the option of describing a characteristic or a function of a product. Fields or functions in a contract can have corresponding attributes in the product. Only contract fields and functions that can be changed need to have corresponding attributes in the product category. Fields and features of a contract specialization that cannot be changed do not need to be created as attributes of the corresponding product category.

The product configurator describes all the IMG activities that must be executed in the IMG under Product Management. The attribute hierarchy consists of all attributes that are available for a product category in Account Management or Master Contract Management.
The attribute hierarchy can consist of the following attribute category:

<table>
<thead>
<tr>
<th>Attribute Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grouping attribute</td>
<td>Grouping attributes group attributes together that belong to the same business topic, and are used for structuring.</td>
</tr>
<tr>
<td>Feature attribute</td>
<td>Feature attributes can be flagged in the attribute hierarchy as required attributes. In the product, they have a padlock that specifies whether the feature can be changed in the contract, and a traffic light that shows the attribute status. Depending on the product settings, a feature can be deactivated in the contract by a feature lock.</td>
</tr>
<tr>
<td>Field Attribute</td>
<td>Field attributes control the field modification, default/fixed values, allowed input values and modification options for one field in one product instance. A field attribute can be a single value or a multiple value attribute.</td>
</tr>
<tr>
<td>Matrix Attribute</td>
<td>Attributes from this category are set up as a matrix and control field values of table fields in the contract according to the two dimensions of the matrix. For eg: Term and Term Unit can form a matrix with respect to a term deposit product.</td>
</tr>
<tr>
<td>List Attribute</td>
<td>A list attribute works in the same way as a matrix attribute that has been defined with one column.</td>
</tr>
</tbody>
</table>

The following function module will retrieve all the valid product attributes for a given product:

```
FUNCTION FSFPR_VA_GET_ALL_Y.
**
** Lokale Schnittstelle:
** IMPORTING
** VALUE(I_PRODINT) LIKE FSFPR_VA_Y_PRODINT
** VALUE(I_VERSION) LIKE FSFPR_VA_Y_VERSION
** VALUE(I_XACTIVE) LIKE FSFPR_VA_Y_XACTIVE
** VALUE(I_CLIENT) LIKE T000-MANDT DEFAULT SY-MANDT
** TABLES
** T_ATTRIBUTE_STRUCTURE TFSFR_ATTRIBUTE_Y_OPTIONAL
** T_ATTRIBUTE_T TFSFR_ATTRIBUTE_T_OPTIONAL
** T_ATTRITREE_STRUCTURE TFSFR_ATTRTREE_Y_OPTIONAL
** T_ATTRIBFIELD_STRUCTURE TFSFR_ATTRIBFLD_Y_OPTIONAL
** T_ATTRIBFUNCTION_STRUCTURE TFSFR_ATTRIBFNCT_Y_OPTIONAL
** T_ATTRIBVALUE_STRUCTURE TFSFR_ATTRIBVAL_Y_OPTIONAL
** T_ATTRIBMATRIX_STRUCTURE TIFPR_ATTRIBMATRIX_OPTIONAL
** T_VA_STRUCTURE FSFPR_VA_Y_OPTIONAL
** T_VA_FLD_STRUCTURE FSFPR_VA_FLD_Y_OPTIONAL
** T_VA_FLM_STRUCTURE FSFPR_VA_FLM_Y_OPTIONAL
** T_VA_FLEDM STRUCTURE FSFPR_VA_FLEDM_Y_OPTIONAL
** T_VA_VALUE_STRUCTURE FSFPR_VA_VALUE_Y_OPTIONAL
** T_VA_MATRIX_STRUCTURE FSFPR_VA_MATRIXOPTIONAL
** T_VA_FLD_SRC_STRUCTURE FSFPR_VA_FLD_SRC_OPTIONAL
** T_VA_FLD_DB STRUCTURE FSFPR_VA_FLD_DB_OPTIONAL
** EXCEPTIONS
** PARAMETER_ERROR
** DATABASE_ERROR
**
```
Once the attributes are retrieved from SAP system, the next biggest challenge is to create the nodes as provided in the XML Schema.

**Definition of XML Schema using Package iXML**

Consider the situation where the customer demands the product attributes need to be extracted as per a predefined XML schema. For eg: You are provided with the following XSD file and you need to extract the product attributes as per the schema.

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">  
  + <xs:element name="ProductList">  
  + <xs:complexType name="ProductType">  
  + <xs:complexType name="VersionType">  
  + <xs:complexType name="VersionAttributeType">  
  + <xs:complexType name="VersionAttributeValidValueType">  
  + <xs:complexType name="VersionMultiAttributeValidValueType">  
  + <xs:complexType name="VersionFieldAttributeType">  
  + <xs:complexType name="ConditionGroupType">  
  + <xs:complexType name="ConditionType">  
  + <xs:complexType name="ConditionItemType">  
</xs:schema>
```

In order to achieve a hierarchy of nodes as given in the xml, schema, we use the package iXML in xml library. The iXML class is the origin from where to start with the iXML library. It serves both as a factory for the elementary objects Document, Parser and StreamFactory and as a "handle" defining the processing context - or session - of subsequent API calls.

```c
* Object References
data: g_iXml = c_iXml->create();
document = g_iXml->create_document();
renderer = g_iXml->create_renderer();
streamFactory = g_iXml->create_stream_factory();
root = document->create_simple_element("ProductList" parent = document); /* this is the root node.*/
```

The create method creates an instance of the iXML class and returns an interface pointer to the instance. The create document method creates a new document instance and returns an interface pointer to the newly created document instance. The iXMLDocument interface represents the entire XML document. Conceptually, it is the root of the document tree, and provides the primary access to the document's data.

In order to map the values correctly into XML elements, a flat structure consisting of all XML elements mentioned in the Schema are created. The elements are created under the root node using the method Create_simple_element. This method creates a simple element with the given name (and namespace) and the specified value as text content. Note that the instance returned implements the iXMLElement interface, so attributes can be specified directly on the returned object.
**Package DOM**

The DOM presents documents as hierarchy of “Node” objects that also implements more specialized interfaces. A node can be termed either as a root node, child node or leaf nodes, which cannot have anything below them in the document structure. The package DOM contains several useful interfaces such as if_ixml_attribute, if_ixml_element, if_ixml_document which can be used for various operations on the document tree.

**Package Stream**

Once the target tree structure is built, the next step is to create the output stream of XML. The stream package contains all definitions to handle XML stream I/O. Input and output of XML documents is handled in terms of XML streams in the iXML library. The stream package defines 3 major concepts: an XML stream factory (iXMLStreamFactory), an XML input stream (iXMLIStream) and an XML output stream (iXMLOStream). The stream factory is used to create XML input and output streams. Since different input sources and output destinations have to be considered, iXMLIStream and iXMLOStream interfaces will be implemented by different classes, each one capable of serving a particular source or destination. Each of these classes is registered or can be registered with the XML stream factory (prototype pattern) and can be queried about its capabilities. This allows the stream factory to create streams for all supported sources or destinations on request in an client application independent way. One of the advantages of this approach is the XML parsers capability to automatically resolve external entity references, as long as there is a stream type registered with the factory, which can handle the protocol defined by the URL.

```java
streamFactory = g_ixml->create_stream_factory().
ostream = streamFactory->create_ostream_itable( xml_table ).
```

XML table is a table of lines with hexadecimal values as shown below.

```plaintext
* type declaration
types: begin of xml_line,
   data(256) type x,
end of xml_line.

* Internal Table
data: xml_table type table of xml_line,
```

The iXMLOStream interface has to be implemented by all XML output streams. Creation of XML output streams is done indirectly by calling the factory method createOSStream() of the client application's StreamFactory instance. The method create_ostream_itable creates a new XML output stream for the given internal table.
Package Renderer

The renderer package contains all definitions required for the XML renderer implemented by the iXML library. For each XML document to be rendered, a separate renderer instance is required. The package Renderer contains the interface if_ixml_renerer. The iXML Renderer interface provides access to the iXML libraries rendering functionality.

```java
renderer = g_ixml->create_renderer( ostream = oStream,
                                    document = document ).
renderer->render( ).
```

This method implements the DOM-based interface to the renderer.
render() returns if the complete DOM-tree has been serialized into the associated output stream

Product Attributes Extract as XML file

Now we are ready with the XML content which can be downloaded as an xml file.

```java
xml_size = oStream->get_num_written_raw( ).
```

```java
CALL METHOD CL_GUI_FRONTEND_SERVICES=>GUI_DOWNLOAD
EXPORTING
   BIN_FILESIZE = xml_size
   FILENAME = filename
   FILETYPE = 'BIN'
CHANGING
   DATA_TAB = xml_table
EXCEPTIONS
   FILE_WRITE_ERROR = 1
   NO_BATCHE = 2
   GUI_REFUSE_FILETRANSFER = 3
   INVALID_TYPE = 4
   NO_AUTHORITY = 5
   UNKNOWN_ERROR = 6
   HEADER_NOT_ALLOWED = 7
   SEPARATOR_NOT_ALLOWED = 8
   FILESZETE_NOT_ALLOWED = 9
   HEADER_TOO_LONG = 10
   DP_ERROR_CREATE = 11
   DP_ERROR_SEND = 12
   DP_ERROR_WRITE = 13
   UNKNOWN_DP_ERROR = 14
   ACCESS_DENIED = 15
   DP_OUT_OF_MEMORY = 16
   DISK_FULL = 17
   DP_TIMEOUT = 18
   FILE_NOT_FOUND = 19
   DATAPROVIDER_EXCEPTION = 20
   CONTROL_FLUSH_ERROR = 21
   NOT_SUPPORTED_BY_GUI = 22
   ERROR_NO_GUI = 23
   others = 24.
```
If the above result needs to be exposed as a web service which will give you an xml output string as a service interface parameter, we need to follow some additional steps as explained below.

**Creation of Enterprise Services from RFC function module**

RFC-enabled function modules, function groups that contain an RFC-enabled function module, BAPIs and XI message interfaces can be made available as Web services without any additional programming. You create a Web service using the Web Service Creation Wizard. In this case, the Web services properties are defined in a preset, selectable profile. To create or consume Web services, we need the authorizations associated with the role **SAP_BC_WEBSERVICE_ADMIN**.

We will try to publish the product attribute extract as a web service so that the product attributes are retrieved as an XML string output by the service consumer. We will write the application logic inside a RFC enabled function module and create a web service from this RFC enabled function module using the Web Service Creation Wizard. The only difference in the application logic here is, instead of streaming the XML output to an internal table, here we are streaming it to a string variable using the Stream Factory.

```c
$stream = streamFactory->create_ostring_cstring( xml_string ).
renderer = g_ixml->create_renderer( oStream = oStream
                                          document = document ).
```

The RFC enabled function module has the following input and output parameters.

<table>
<thead>
<tr>
<th>Function module</th>
<th>Z_COMP_PRODCT_EXTRACT</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attributes</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Import</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Export</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Changing</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tables</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Exceptions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Source code</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```c
FUNCTION Z_COMP_PRODCT_EXTRACT.
***Local Interface:
** IMPORTING
** VALUE(PRODINT) TYPE FSPR_PRODINT_Y OPTIONAL
** VALUE(PRODEXT) TYPE FSPR_PRODEXT_Y OPTIONAL
** VALUE(PRODTYPE) TYPE FSPR_PRODTYPE_Y OPTIONAL
** EXPORTING
** VALUE(XML_STRING) TYPE STRING
```

A web service is created from the above function module using Web Service Creation Wizard.
The XML content is available in the output parameter XML_String.
Results

The above illustrations clearly states how the product attributes can be extracted into a pre defined XML schema using the iXML library and Web Services.

The extracted product attributes output in XML is as shown below.

```xml
<?xml version="1.0" ?>
  <ProductList>
    - <Product>
      <Client>200</Client>
      <ProductInternalID>I0000000</ProductInternalID>
      <ProductExternalID>Z001</ProductExternalID>
      <ProductCategoryCode>0002</ProductCategoryCode>
      <ProductDescription>Streamline Account</ProductDescription>
      <DateCreated>2008-09-01T00:00:00</DateCreated>
      <LogicalSourceSystem>ZB1CLNT200</LogicalSourceSystem>
      <TimeCreated>10:56:28</TimeCreated>
    - <ProductVersion>
      <ProductVersion>0015</ProductVersion>
      <AllowProductInstanceId>true</AllowProductInstanceId>
      <ProductVersionStatus>30</ProductVersionStatus>
      <ValidFrom>2000-01-01T00:00:00</ValidFrom>
      <ValidTo>9999-12-31T00:00:00</ValidTo>
      <DateCreated>2008-09-01T00:00:00</DateCreated>
      <TimeCreated>10:56:28</TimeCreated>
      <CreatedBy>BUTTI3R</CreatedBy>
      <DateChanged>2008-09-11T00:00:00</DateChanged>
      <TimeChanged>10:40:26</TimeChanged>
      <ChangedBy>BUTTI3R</ChangedBy>
      <DateReleased>2008-09-11T00:00:00</DateReleased>
      <TimeReleased>10:29:44</TimeReleased>
      <ReleasedBy>BUTTI3R</ReleasedBy>
    - <VersionAttribute>
      <AttributeID>SAP AM 000048</AttributeID>
      <Description>Description of Condition</Description>
      <AttributeStatus>1</AttributeStatus>
      <PadlockStatus>false</PadlockStatus>
    + <VersionFieldAttribute>
    - <VersionAttribute>
      <AttributeID>SAP AM 000049</AttributeID>
      <Description>Valid To</Description>
      <AttributeStatus>1</AttributeStatus>
    </VersionAttribute>
  - <VersionAttribute>
```

© 2008 SAP AG
Related Content

http://help.sap.com/saphelp_dm40/helpdata/en/75/eb853c5bd34414e10000000a11405d/frameset.htm
http://help.sap.com/saphelp_nw04/helpdata/EN/86/8280ba12d511d5991b00508b6b8b11/frameset.htm
http://help.sap.com/saphelp_nw70/helpdata/EN/9b/dad1ae3908ee44a5caf57e10918be9/frameset.htm

For more information, visit the Web Services homepage.