Business Objects

Business Objects are the data abstractions used in WPS. When data is received or sent between components or services, if the data is anything other than simple data, it is treated as a Business Object. Think of the Business Object as a logical container of fields where each field has a name and a data type. The data type can be a simple data type such as String or Integer or can itself be another Business Object.

SDO Programming Model v1.0

Business Objects are SDO. The in memory representation of a business object is a 1.0 Service Data Object. The programming model is the SDO 1.0 programming model as a base with the Business Object Framework Services providing incremental services in addition to what is available in SDO 1.0. It is recommended that you leveragethe 1.0 Java Doc for reference. It is available for download from developerWorks here:

SDO Programming Model v2.0

The WPS Business Object Framework is not based on this version but you can access the published 2.0 doc at this address.

Recording changes

One of the primary functions of SDO has been for the 'disconnected data source' pattern. This model is the idea that a data owning system can provide a copy of an instance of data that it owns and that data is contained within the SDO. The application that requested the data can now work against the SDO copy of the data. At some time in the future, the SDO DataObject can be supplied back to the data owning system and, if possible, only the changed data items need to be updated by that system. This disconnected data model allows a consumer of data to get a copy, be disconnected from the originating data system, do work against that data and then have the resulting changes pushed back into the data owning system. SDO attempted to provide

this function through a mechanism it called a DataGraph. The reality of the situation after years of SDO availability and practice is that this capability is never used.

Splitting namespaces across projects

Be very very careful when splitting namespaces across projects. The following won't work (by design)

MyLibrary contains BO1 in namespace http://com.test.types

MyModule contains Iface1 and BO2 in namespace http://com.test.types

BO1 will not be visible to Iface1 or BO2 (or anything else in MyModule with namespace http://com.test.types). From the editor - you will not even be able to create an attribute in BO2 of type BO1. Nor an input/output parameter of type BO1 in Iface1.

Best practice: do not split namespaces across projects.

What you should be aware of: This applies to null namespaces.

Business Object Programming

This section describes some of the issues and concepts relating to Business Object programming in Java.

SDO Representation of XML Schema Types

SDO Representation of XML Schema Types

Arrays

When a Business Object field is flagged as an Array, the DataObject getter and setter is getList() and setList() which interact with the java.util.List class.

Note that getList() will always return a List object, never null. The returned List can track changes to the Business Object (it is not just an ArrayList or LinkedList), which is important for Business

Objects that use an internal Sequence (see below).

Similarly setList() does not replace the internal List, it replaces the contents (according to the SDO 1.0 spec, it is equivalent to clear() followed by addAll().

The List interface has a size() method that can be used to determine the number of elements in the list (and hence the number of elements in the array).

Sequenced Objects

When a Business Object represents complex content (for example, nested, repeating sequences in XML Schema), the object maintains an internal sequence. As properties are set, or values added to Array properties (Java List interface), the internal sequence is updated automatically. For these complex cases, it is the responsibility of the programmer to set / add values in an order that matches the schema.

Nested BOs

BOs typically can contain other BOs as fields resulting in a tree or hierachical structure. To create a child BO, one could use the BOFactory and create an appropriately typed BO and then add it but there is an easier way. The DataObject contains a method called createDataObject that takes a parameter a property name and returns a DataObject of the appropriate type for the named property contained in the parent DataObject.

For example if a BO called "A" contains:

---

then if we have a reference to an "A" DataObject calling the A.createDataObject("Billing") method will return a DataObject of type Address.

BOFactory: createByElement or create?

In WPS 6.0.1+ there are two methods on the BOFactory you will get to know well. They are createByElement and create. Use create when the Business Object xsd definition is contained in a <xsd:complexType> tag. Use createByElement when the Business Object xsd definition is contained in a <xsd:element> tag. If you created your Business Object in WID, then it generated an xsd that via the complexType tag and you will use the create(..) method.

BOs of 'any' type

Firstly, there are (at least) 3 cases to be aware of:

In the first case, a BO can specify the base type for a property, and at runtime a BO inheriting from the base can be used. This is supported in the runtime and the BO editor.

In the second case, the corresponding property will have a type of Object (not DataObject) at runtime. The BO editor will let you specify this, however it could cause problems at runtime (where components assume that they will get a DataObject, not an Object).

In the third case, this is a schema feature and shouldn't be used in BOs, but may occur where an external schema is imported (e.g. Web Services). SDO supports this for parsing XML / reading from the DataObject, but not when creating a DataObject. See technote 1250444 'Problems associated with using xsd:any' for a workaround.

WSDL files and externally referenced XSDs

WID

When dealing with remote business objects, the current situation (PMR 90542-999-000) requires you to import the XSDs into WID. You can do this by

* File > Import... > HTTP & then specify the source & the target locations.

By importing the XSDs, the other editors are able to use them.

WPS

If you currently want to reference a "remote BO," defined as one which has the schemaLocation attribute pointing to a URI that contains the XSD, the runtime will load it. If you cannot load your "remote" BO, check that the schemaLocation attribute is set properly (for example, you are able to load get the XSD from the URI you specified) and then open a PMR. In some future release of WPS, there may be truely "remote" BOs in which you do not need to hardcode the schema location.

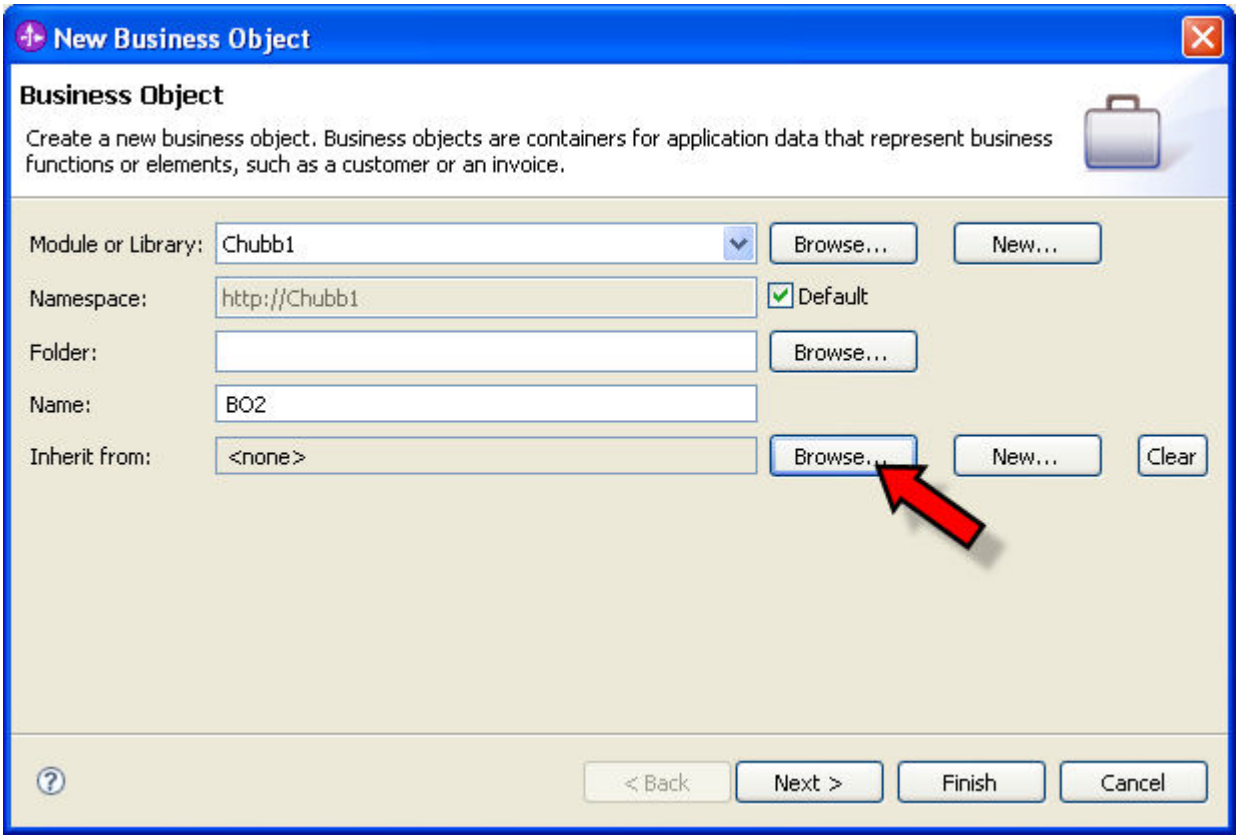
OpenSource SDO Utilities and Classes

As we build SDO based projects, we will find that we generate code artifacts that do interesting things (such as dump the contents of a DataObject). To capture such artifacts, an Open Source (but internal to IBM) repository has been created that is built on a CVS server. This allows you to immediately import useful function directly into WID. You can also update the code or add new assets and post them back to the repository. There are more details available on the OpenSource repository.

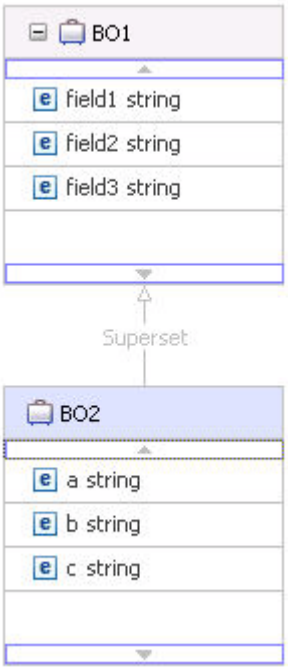
Business Object Inheritance

A Business Object can inherit from another business object when it is defined. This allows one BO to be a superset of another. The superset BO contains all of the fields of the subset BO plus additional fields. In an interface, a subset BO specified as a parameter can be passed a superset BO as an actual parameter.

The following shows the creation of a new Business Object called BO2. We want this to inherit from BO1 and select BO1 in the "Inherit from" area.



The result is a new BO called BO2 that is a superset of BO1.



Application Specific Information (ASI)

Business Objects can have meta data associated with them and the fields contained within them. This meta data is called "Application Specific Information" or "ASI". Unfortunately, there does not appear to be any SDO API to retrieve (or set) this information. Fortunately IBM has provided a class called AdapterBOUtil that is supplied with the Adapter Foundation Class (CWYBS_AdapterFoundation.jar. There are a number of methods for working with the ASI information including:

See Also:

DeveloperWorks – Validating business objects in Websphere Process Server – 2009-07-29

SDO Programming

The Service Data Object (SDO) has a rich set of programming APIs. JavaDoc for the APIs (v2.4.2) is available as well as more general documentation.

This section shows some illustrative techniques when working with the SDO API.

Walking a DataObject

Walking or iterating through a DataObject can be achieved through the following code fragment.

DataObject myDO; ... Type myType = myDO.getType(); List propertiesList = myType.getProperties(); Iterator i = propertiesList.iterator(); while(i.hasNext()) { Property currentProperty = (Property)i.next(); ... Object value = myDo.get(currentProperty); }

The code works by retrieving the Type of the DataObject. The type contains details of all the different properties contained within. A Java list of those properties is obtained. Next, the list is iterated and something done with each of the properties within.

Create and Populate a DataObject

This method creates a DataObject of the named type. An optional (specify null if not needed) map is supplied containing name/value pairs. For each name, a field in the DataObject is expected to exist that can be set to the supplied value.

public DataObject createDataObjectWithContent(String namespace, String name, HashMap<String, Object> map) { BOFactory boFactory = (BOFactory) ServiceManager.INSTANCE .locateService("com/ibm/websphere/bo/BOFactory"); DataObject dataObj = boFactory.create(namespace, name); if (dataObj == null) { return null; } if (map == null) { return dataObj; } Set<String> keySet = map.keySet(); Iterator<String> i = keySet.iterator(); while (i.hasNext()) { String key = i.next(); Object value = map.get(key); dataObj.set(key, value); } return dataObj; }

IBM DataObjectUtils

IBM provides a class called com.ibm.bpc.clientcore.DataObjectUtils which parses a DataObject and populates a HashMap with the content of that DataObject. The keys to the hashmap are the XPath strings that would be used to access a field. The values of map are the contents of the object. In addition to parsing an object, the map can also be re-written into the object.

This class was designed for JSF usage and can be located in the JAR:

<WPS>/ProcessChoreographer/client/bpcjsfcomponents.jar

Testing seems to show that the class only works in a JSF environment which is a pity because it seems generically useful.

SCA Business Objects

Creating an SCA Business Object

A Business Object represents a structured piece of data. The Business Object can be created in an SCA application through a Java class called the BOFactory. This factory must be retrieved from a named SCA location:

```
BOFactory boFactory =
(BOFactory)ServiceManager.INSTANCE.locateService("com/ibm/websphere/bo/BOFactory");
```

The BOFactory is part of the com.ibm.websphere.bo package and is fully documented along with the other related classes. At the simplest level, to create a new Business Object, we can use:

```
DataObject dataObj = boFactory.create("targetNameSpace", "complexTypeName");
```

As you will notice, the DataObject is created from the factory through the combination of target namespace and type name. The BOFactory appears to scan its current class path looking for XML or XSD documents that match these requirements and, from these values, then creates the associated DataObject. What this means is that if you need to create a DataObject in an arbitrary class, then the BO (.xsd file describing it) needs to be included as a dependency in the project in which the calling class is contained.

Here is an example of creating a DataObject. If you have a WPS Library called "Lib" that contains a BO definition called "Customer" in namespace "http://mybiz", then if you want to create a DataObject instance of that type you would need to call:

```
DataObject myObject = boFactory.create("http://mybiz", "Customer");
```

If the code that creates the BO is in a WPS Module, then "Lib" must be flagged on that Module as a dependency.

If the code that creates the BO is a J2EE App, then the JAR file representing "Lib" must be added as a dependency to the J2EE Deployment Descriptor.

Converting to/from XML

WPS provides a utility class to convert a DataObject to/from a specific formatted XML document. The class implementing this function is called BOXMLSerializer. An instance of this class can be retrieved through:

BOXMLSerializer mySerializer = (BOXMLSerializer) ServiceManager.INSTANCE.locateService("com/ibm/websphere/bo/BOXMLSerializer");

Some of the methods of this class expect a rootElementName. To determine the root of a current DataObject, use the following:

dataObject.getType().getName()

Some of the methods of this call expect a namespace. To determine the namespace of the current DataObject, use the following:

dataObject.getType().getURI();

The following example illustrates turning a BO into an XML document:

BOXMLSerializer mySerializer = (BOXMLSerializer) ServiceManager.INSTANCE.locateService("com/ibm/websphere/bo/BOXMLSerializer"); String rootElementName = dataObject.getType().getName(); String targetNamespace = dataObject.getType().getURI(); ByteArrayOutputStream baos = new ByteArrayOutputStream(); mySerializer.writeDataObject(dataObject, targetNamespace, rootElementName, baos); String xmlText = new String(baos.toString());

To convert an XML string to a DataObject, the following code may be utilized:

String xmlText = "<XML to become a BO>"; ... BOXMLSerializer mySerializer = (BOXMLSerializer) ServiceManager.INSTANCE.locateService("com/ibm/websphere/bo/BOXMLSerializer"); ByteArrayInputStream bais = new ByteArrayInputStream(xmlText.getBytes()); BOXMLDocument document = mySerializer.readXMLDocument(bais); DataObject dataObject = document.getDataObject();

XML documents with no namespace

It is common to find XML documents that have no namespace information associated with them. In order to be able to support this kind of document, the corresponding Business Object should also be set to have no namespace associated with it.

When WPS is supplied an XML document to be turned into a Business Object, it examines that document and provides a fully qualified name of the format {Namespace}Name. This is used to then find the corresponding Business Object definition.

When an XML document with no namespace is presented, then the result is {}Name. In order to find a matching Business Object definition, the BO must also appear as {}Name and this is achieved through nulling out the namespace for the BO. This will result in a WID based warning but this may be safely ignored.

Here is an example. Consider a Business Object called BO1 (no namespace) that has three fields. Converting this to its XML representation results in:

<?xml version="1.0" encoding="UTF-8"?> <BO1 xsi:type="BO1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <f1>1</f1> <f2>2</f2> <f3>3</f3> </BO1>

Revision #1

Created 2 years ago by Admin

Updated 2 years ago by Admin