HPSA Extension Pack

InventoryBuilder Webservice User Reference

Release v.2.4

Legal Notices

Table of Contents

Support

Support for the HP Service Activator Extended Pack product is available on the following mailing list:

ovsa.spain.support@hp.com

In This Guide

This guide is meant as a user reference guide for the Inventory Builder's webservice latest version. It contains all the information about this tool, its features and how to use them.

Audience

The audience for this guide is the Solutions Integrator (SI). The SI has a combination of some or all of the following capabilities:

Understands and has a solid working knowledge of:

- UNIX® commands
- Windows® system administration

Understands networking concepts and language

Is able to program in Java™ and XML

Understands security issues

Understands the customer's problem domain

References

Manual Organization

This guide contains the following chapters:

Chapter 1, "Introducction",

Chapter 2, "General description",

Conventions

The following typographical conventions are used in this guide.

Font	What the Font Represents	Example
Italic	Book or manual titles, and man page names	Refer to the *HP Service Activator — Workflows and the Workflow Manager* and the *Javadocs* man page for more information.
	Provides emphasis	You *must* follow these steps.
	Specifies a variable that you must supply when entering a command	Run the command: `InventoryBuilder` *<sourceFiles>*
	Parameters to a method	The *assigned_criteria* parameter returns an ACSE response.
Bold	New terms	The **distinguishing attribute** of this class...
`Computer`	Text and items on the computer screen	The system replies: `Press Enter`
	Command names	Use the `InventoryBuilder` command ...
	Method names	The `get_all_replies()` method does the following...
	File and directory names	Edit the file `$ACTIVATOR_ETC/config/mwfm.xml`
	Process names	Check to see if `mwfm` is running.
	Window/dialog box names	In the `Test and Track` dialog...
	XML tag references	Use the `<DBTable>` tag to...
`Computer Bold`	Text that you must type	At the prompt, type: **`ls -l`**
Keycap	Keyboard keys	Press **Return**.
[Button]	Buttons on the user interface	Click `[Delete]`. Click the `[Apply]` button.
Menu Items	A menu name followed by a colon (:) means that you select the menu, then the item. When the item is followed by an arrow (->), a cascading menu follows	Select Locate:Objects->by Comment.

Install Location Descriptors

The following names are used throughout this guide to define install locations.

Descriptor	What the Descriptor Represents
`$ACTIVATOR_OPT`	The install base location of Service Activator. The UNIX location is `/opt/OV/ServiceActivator` The Windows location is `<drive>:\HP\OpenView\ServiceActivator\`
`$ACTIVATOR_ETC`	The install location of specific Service Activator configuration files. The UNIX location is `/etc/opt/OV/ServiceActivator` The Windows location is `<drive>:\HP\OpenView\ServiceActivator\etc\`
`$ACTIVATOR_VAR`	The install location of specific Service Activator logging files. The UNIX location is `/var/opt/OV/ServiceActivator` The Windows location is `<drive>:\HP\OpenView\ServiceActivator\var\`
`$ACTIVATOR_BIN`	The install location of specific Service Activator binary files. The UNIX location is `/opt/OV/ServiceActivator/bin` The Windows location is `<drive>:\HP\OpenView\ServiceActivator\bin\`
`$ACTIVATOR_THIRD_PARTY`	The location for new Java components such as workflow nodes and modules. Third-party libraries can also be placed in this directory. The UNIX location is `/opt/OV/ServiceActivator/3rd-party` The Windows location is `<drive>:\HP\OpenView\ServiceActivator\3rd-party\` Customized inventory files are stored in the following locations: UNIX: `$ACTIVATOR_THIRD_PARTY/inventory` Windows: `$ACTIVATOR_THIRD_PARTY\inventory`
`$JBOSS_HOME`	HOME The install location for JBoss. The UNIX location is `/opt/HP/jboss` The Windows location is `<drive>:\HP\jboss`
`$JBOSS_DEPLOY`	The install location of the Service Activator J2EE components. The UNIX location is `/opt/HP/jboss/server/default/deploy`

	The Windows location is `<drive>:\HP\jboss\server\default\deploy`
`$ACTIVATOR_DB_USER`	The database user name you define. Suggestion: `ovactivator`
`$ACTIVATOR_SSH_USER`	The Secure Shell user name you define. Suggestion: `ovactusr`
`$SOSA_HOME`	The install base location of SOSA. The default UNIX location is `/opt/OV/Sosa` The default Windows location is <drive>:\HP\OpenView\Sosa\
`$SOSA_BIN`	The install location of specific SOSA binary files. The default UNIX location is `/opt/OV/Sosa/bin` The default Windows location is `<drive>:\HP\OpenView\Sosa\bin\`
`$SOSA_ETC`	The install location of specific SOSA configuration files. The default UNIX location is `/opt/OV/Sosa/config` The default Windows location is <drive>:\HP\OpenView\Sosa\config\
`$ECP_HOME`	The install base location of Equipment Connections Pool. The default UNIX location is `/opt/OV/ECP` The default Windows location is <drive>:\HP\OpenView\ECP\
`$ECP_BIN`	The install location of specific Equipment Connections Pool binary files. The default UNIX location is `/opt/OV/ECP/bin` The default Windows location is <drive>:\HP\OpenView\ECP\bin\
`$ECP_ETC`	The install location of specific Equipment Connections Pool configuration files. The default UNIX location is `/opt/OV/ECP/conf` The default Windows location is <drive>:\HP\OpenView\ECP\conf\

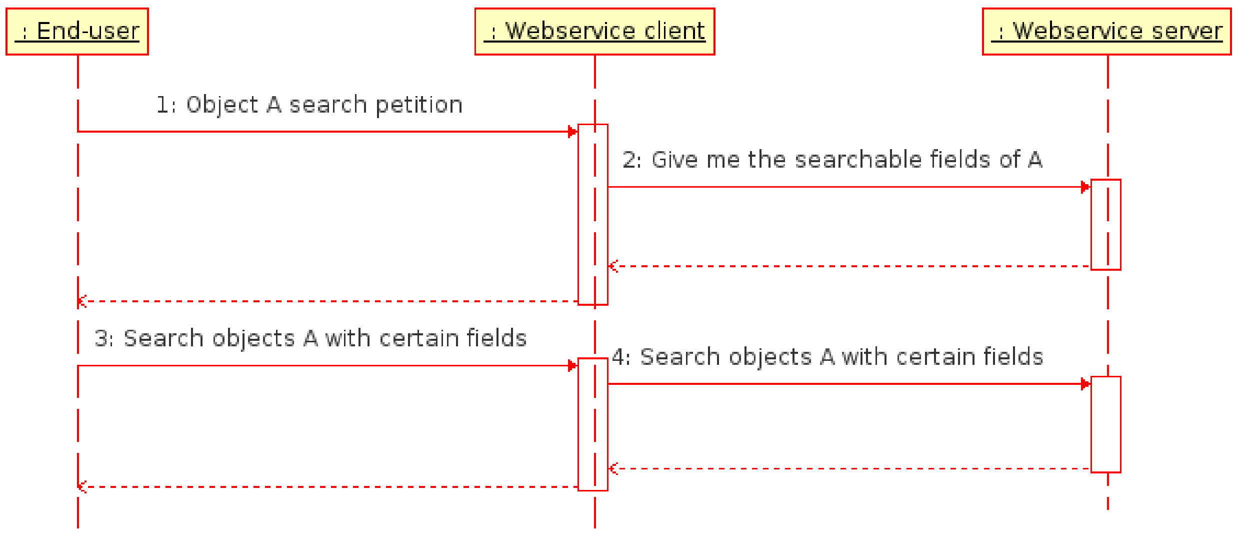
1 General description

The OVSA SPI application provides a Web service (implemented trough Axis2 1.4.1) called BeanService. Trough this webservice the user can consult the structure of any of the beans created by the Inventory Builder and access to the search, store, update and delete actions. These beans must be in the class path.

The axis2 web application is located in the diagnostic server (port 8089 by default) in the path "/axis2/services/BeanService".

A copy of the wsdl can be accessed by calling
`http://host:8089/axis2/services/BeanService?wsdl`

The idea is that the webservice client does not need to know the definition of the bean to operate with it. The diagram illustrating a search sequence is showed below.


: End-user
: Webservice client
: Webservice server
1: Object A search petition
2: Give me the searchable fields of A
3: Search objects A with certain fields
4: Search objects A with certain fields

The other operations are analog.

2 Functionality

The remote interface of the BeanService is defined in the BeanService.wsdl and service.xml that can be found in the BeanService.aar. These methods provide some functionality that will be documented in detail in the next chapters:

2.1 Fields discovery

As told before, the idea is that the webservice client does not need to know the definition of the bean to operate with it. To archive this some methods must be provide to recover the information about the field. These fields will be presented to the user (with a description) and he must fill the necessary values for the operation.

2.1.1 getAllFields

This method returns the information of all the fields of the Bean.

2.1.1.1 Input parameters

- bean_name.- Name of the bean to recover the information.

2.1.1.2 Output parameters

The return is an object (named FieldDescriptor) with these fields:

- askToTheUserBehavior.- Informs about the way to show up a field to the user when asking him to introduce a value.
- canBeUsedInSearches.- Indicates if the field can be used for searching.
- dateFormat.- Format used to represent a date (if the field is a date)
- description.- Human readable text that describes the field.
- mandatory.- Indicates if the field must be present when storing a new object.
- name.- Name of the field.
- showToTheUser.- Indicates if the field has to be showed up to the user in a search result.

- toBeShadowed.- Indicates if the content of the text field (when asking to the user) must be changed with * so no one can see what's typing.
- type.- Data type.
- value.- Default value of the field.

2.1.2 getStorableFields

This method returns the description of the fields that can be accepted by the store method. Note that the output parameter *mandatory* will inform about the optional characteristic of that field. This way the presence or not of the fields depends on two parameters:

getStorableFields	mandatory	result
Yes	Yes	Has to be present
Yes	No	No need to be present
No	XXX	Cannot be present

2.1.2.1 Input parameters

- bean_name.- Name of the bean to recover the information.

2.1.2.2 Output parameters

The return is an object (named FieldDescriptor) with these fields:

- askToTheUserBehavior.- Informs about the way to show up a field to the user when asking him to introduce a value.
- canBeUsedInSearches.- Indicates if the field can be used for searching.
- dateFormat.- Format used to represent a date (if the field is a date)
- description.- Human readable text that describes the field.
- mandatory.- Indicates if the field must be present when storing a new object.
- name.- Name of the field.

- showToTheUser.- Indicates if the field has to be showed up to the user in a search result.
- toBeShadowed.- Indicates if the content of the text field (when asking to the user) must be changed with * so no one can see what's typing.
- type.- Data type.
- value.- Default value of the field.

2.1.3 getSearchableFields

This method returns the description of the fields that can be used to do a search. The result is exactly the same that if the user do a getAllFields and remove all the fields with *canBeUsedInSearches == false*.

2.1.3.1 Input parameters

- bean_name.- Name of the bean to recover the information.

2.1.3.2 Output parameters

The return is an object (named FieldDescriptor) with these fields:

- askToTheUserBehavior.- Informs about the way to show up a field to the user when asking him to introduce a value.
- canBeUsedInSearches.- Indicates if the field can be used for searching.
- dateFormat.- Format used to represent a date (if the field is a date)
- description.- Human readable text that describes the field.
- mandatory.- Indicates if the field must be present when storing a new object.
- name.- Name of the field.
- showToTheUser.- Indicates if the field has to be showed up to the user in a search result.
- toBeShadowed.- Indicates if the content of the text field (when asking to the user) must be changed with * so no one can see what's typing.
- type.- Data type.
- value.- Default value of the field.

2.2 Store a new bean

To store a new bean, the user must use the store method.

2.2.1 store

This method stores a bean in the database.

2.2.1.1 Input parameters

- bean_name.- Name of the bean to recover the information.
- Array of fields ([0..*]).- Data of the new bean. These fields must be present when invoking getStorableFields. And at least all the mandatory ones have to be present.

2.2.1.2 Output parameters

The return is a boolean that indicates if the store operation has been ok.

2.3 Update an existent bean

To update a bean, the user must use the update method.

2.3.1 update

This method updates a bean in the database. The primary keys cannot be updated.

2.3.1.1 Input parameters

- bean_name.- Name of the bean to recover the information.
- primaryKey.- Value of the primary key of the bean that wants to be updated.

- Array of fields ([0..*]).- Data of the bean to be updated. There cannot be a field part of the primary key.

2.3.1.2 Output parameters

The return is a boolean that indicates if the update operation has been ok.

2.4 Delete an existing bean

To delete a bean, the user must use the delete method.

2.4.1 delete

This method removes a bean in the database.

2.4.1.1 Input parameters

- bean_name.- Name of the bean to recover the information.
- primaryKey.- Value of the primary key of the bean that wants to be updated.

2.4.1.2 Output parameters

The return is a boolean that indicates if the delete operation has been ok.

2.5 Searching for a bean

To search for beans, the user must use the search method.

2.5.1 search

Search for a certain beans in the database. Returns the data of the beans en the description of the fields.

2.5.1.1 Input parameters

- bean_name.- Name of the bean to recover the information.
- Array of field_searchs ([0..*]).- Data of the bean to be searched by. These fields must be present when invoking getSearchableFields.
- advanced.- Indicates if the normal or advanced version of the search must be used.
- max_result.- Maximum number of result to return.

2.5.1.2 Output parameters

The return is an object named BeanReturnType. In it an array of String called *fieldNames* can be found. This array contains the name of the fields, the rest arrays represents the values for the field in the same position as the fieldNames. For example, if fieldNames[2] = "Telephone", then to find the description of that field we have to look in the fieldDescriptions[2].

Because the field descriptors are the same, there only need to send them one time. But with the values is different. The response contains all the values. So the fieldValues is a array of arrays. This way the fieldValues[1] is the second bean found in the database and the fieldValues[1][2] is the telephone number.

The fields of the object BeanReturnType are:

- askToTheUserBehaviors. - Array of askToTheUserBehavior ([0..*])
- fieldDateFormats. - Array of dateFormat ([0..*])
- fieldDescriptions. - Array of description ([0..*])
- fieldNames. - Array of name ([0..*])
- fieldTypes. - Array of type ([0..*])
- fieldValues. - Array of BeanValues ([0..*])
- Array of mandatory ([0..*])
- passwords. - Array of description ([0..*])
- searchables. - Array of description ([0..*])
- Array of showToTheUser ([0..*])

3 Client example

Generating the client side with the **Apache Axis2 ADB** method we can implement a class that connects to the webservice methods like the following one:

```
import java.rmi.RemoteException;

import org.apache.axis2.AxisFault;

import test.com.hp.ov.activador.inventorybuilder.constants.IBTestProperties;
import test.com.hp.spain.spi.client_webservices.types.BeanServiceStub;
import
test.com.hp.spain.spi.client_webservices.types.BeanServiceStub.FieldStore;

import com.hp.ov.activator.inventorybuilder.utils.Log;



/**
 * BeanService Client
 *
 * @author Mario CerdeÃ±o
 */
public class RPCBeanServiceClient {

  private static BeanServiceStub stub;

  static {

      try {
            stub = new
BeanServiceStub("http://localhost:8089/axis2/services/BeanService");
      } catch (AxisFault e) {
            e.printStackTrace();
            Log.error("Can not inizializate the service. Is the jboss up in
" + IBTestProperties.WEB_SERVICE_URL + "?");
```

```
      }

  }



  /**
   * Stores a bean through the webservice.
   *
   * @param bean_name
   *          package and name of the bean
   * @param fields_store
   *          fields to store.
   * @return true if the storation has been ok
   * @throws RemoteException
   */
  public static boolean store(String bean_name, BeanServiceStub.FieldStore[]
fields_store) throws RemoteException {


      BeanServiceStub.Store store = new BeanServiceStub.Store();
      store.setBean_name(bean_name);
      store.setFields(fields_store);
      return ((BeanServiceStub.StoreResponse)
stub.store(store)).get_return();


  }



  /**
   * Deletes a bean through the webservice.
   *
   * @param bean_name
   *          package and name of the bean
   * @param primaryKey
   *          primary key of the bean instance to delete
   * @return true if the deletion has been ok
   * @throws RemoteException
   */
  public static boolean delete(String bean_name, String primaryKey) throws
RemoteException {
      BeanServiceStub.Delete delete = new BeanServiceStub.Delete();
      delete.setBean_name(bean_name);
      delete.setPrimaryKey(primaryKey);

            return ((BeanServiceStub.DeleteResponse)
stub.delete(delete)).get_return();

      }

  /**
   * Updates a bean through the webservice
   *
   * @param bean_name
   *          package and name of the bean
   * @param primaryKey
   *          primary key of the bean instance to update
   * @param fields
   *          fields to update
   * @return true if the update has been ok
   * @throws RemoteException
   */
  public static boolean update(String bean_name, String primaryKey,
FieldStore[] fields) throws RemoteException {

      BeanServiceStub.Update update = new BeanServiceStub.Update();
      update.setBean_name(bean_name);
      update.setFields(fields);
      update.setPrimaryKey(primaryKey);

      return ((BeanServiceStub.UpdateResponse)
stub.update(update)).get_return();

  }

  /**
   * Returns all the fields of a bean
   *
   * @param bean_name
   *          package and name of the bean
   * @return all the fields of a bean
   * @throws RemoteException
   */
  public static BeanServiceStub.GetAllFieldsResponse getAllFields(String
bean_name) throws RemoteException {

      BeanServiceStub.GetAllFields fields = new
BeanServiceStub.GetAllFields();
      fields.setBean_name(bean_name);

      return stub.getAllFields(fields);

  }

  /**
   * Returns the storable fields of a bean
   *
   * @param bean_name
   *          package and name of the bean
   * @return the storable fields of a bean
   * @throws RemoteException
   */
  public static BeanServiceStub.GetStorableFieldsResponse
getStorableFields(String bean_name) throws RemoteException {

      BeanServiceStub.GetStorableFields fields = new
BeanServiceStub.GetStorableFields();
      fields.setBean_name(bean_name);

      return stub.getStorableFields(fields);

  }



  /**
   * Returns the searchable fields of a bean
   *
   * @param bean_name
   *          package and name of the bean
   * @return the searchable fields of a bean
   * @throws RemoteException
   */
  public static BeanServiceStub.GetSearchableFieldsResponse
  getSearchableFields(String bean_name) throws RemoteException {


      BeanServiceStub.GetSearchableFields fields = new
BeanServiceStub.GetSearchableFields();
      fields.setBean_name(bean_name);


      return stub.getSearchableFields(fields);


  }



  /**
   * Performs a search trough a webservice
   * @param bean_name
   * @param search_fields
   * @param advanced
   * @param max_results
   * @return
   * @throws RemoteException
   */
  public static BeanServiceStub.SearchResponse search(String bean_name,
BeanServiceStub.FieldSearch[] search_fields
                                                 , boolean advanced, int
max_results) throws RemoteException {

      BeanServiceStub.Search search = new BeanServiceStub.Search();
      search.setAdvanced(advanced);
      search.setBean_name(bean_name);
      search.setMax_results(max_results);
      if (search_fields != null) search.setField_searchs(search_fields);
      return stub.search(search);


  }




}
```

4 Parameters descriptions

4.1 Input parameters

4.1.1 bean_name

String with the package and name of the bean, for example "*com.hp.spain.inventory.Net*".

4.1.2 fields

Object to pass fields with its values as parameters. It is composed by:

- name.- Name of the field
- value.- Value of the field

4.1.3 primaryKey

It's the string representing the primary key of a bean. For simple keys is just the key itself, for compose keys is a concatenation of all the keys with the char "|". The user doesn't need to implement that because any object returned with the search method has a field *primaryKey* with exactly this value.

4.1.4 advanced

boolean that indicates if the search must be normal or advanced. In other words, false return the data has found in the database table, true change the ListOfValues fields for the value pointed. See the Inventory Builder User Reference for more info about this.

4.1.5 max_results

int with the maximum number of results permitted. An Exception is thrown when more results. This is useful for avoid charge to the database.

4.1.6 field_search

Object to pass fields to search for. It is composed by:

- name.- Name of the field
- operation. - SQL comparison for this field. It's a literal, one of:
  - **IN**. - The sql operation will be *Field IN (?,?)* with as many ? as values
  - **LIKE**. - The sql operation will be *Field LIKE '%'?'%'*.
  - **STARTS_WITH**. - The sql operation will be *Field LIKE ?'%'*.
  - **ENDS_WITH**. - The sql operation will be *Field LIKE '%'?*.
  - **BETWEEN**. - The sql operation will be *Field BETWEEN ? AND ?*. The number of values must be two.
  - **EQUAL**. - The sql operation will be *Field = ?*. If the value is null the sql operation will be "Field IS NULL".
  - **NOT_EQUAL**. - The sql operation will be *Field <> ?*. If the value is null the sql operation will be *Field IS NOT NULL*.
  - **LESS**. - The sql operation will be *Field < ?*.
  - **EQUAL_OR_LESS**. - The sql operation will be *Field <= ?*.
  - **MORE**. - The sql operation will be *Field > ?*.
  - **EQUAL_OR_MORE**. - The sql operation will be *Field >= ?*.
- values. - Array of strings with the values to be compared.

4.2 Output parameters

4.2.1 askToTheUserBehavior

String with a predefined behavior that informs about the way to show up a field to the user when asking him to introduce a value (for example when storing a bean). The values can be (case sensitive) one of:

- **Normal**. -The value has to be filled by the user.
- **Optional**. - The user doesn't need to fill this value.
- **Read Only**. - The user can read the value that will be assigned but cannot change it.

- **Not ask**. - The value does not be showed up to the user.

4.2.2 canBeUsedInSearches

boolean that indicates if the field can be used for searching.

4.2.3 dateFormat

String with the date format (java style) of the field if this one is a date. If not, the value will be null.

4.2.4 description

String with a description of the field that can be used to show up to the user.

4.2.5 mandatory

boolean that indicates if the field is mandatory (have to be passed by when storing).

4.2.6 name

String with the name of the field.

4.2.7 showToTheUser

boolean that indicates if the field has to be showed up to the user in a search result.

4.2.8 toBeShadowed

boolean that indicates if the form field text has to be shadowed (for example changing it by *) when asking to the user for it. An example of this behavior is when asking about a password.

4.2.9 type

String with a predefined type, the type of the field. Can be (case sensitive) one of:

- **int**
- **long**
- **float**
- **double**
- **boolean**
- **String**
- **Date**
- **Blob**
- **Clob**

4.2.10 value

String with the actual value of the field. If is part of the response to a non-search action (getAllFields for example) then indicates the default value of the field.

4.2.11 BeanValues

Object with an array of the values of the bean's fields. In the same response there will be an array of field names that indicates which is the order of this values (the same that the field names).