HP Select Identity

Software Version: 4.20

External Call Developer Guide

Document Release Date: September 2007
Software Release Date: September 2007

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

Trademark Notices

UNIX® is a registered trademark of The Open Group.

The Select Identity product CD contains a `license` directory where you can find the license agreements for each of the third-party products used in this product.

Support

You can visit the HP software support web site at:

www.hp.com/go/hpsoftwaresupport

HP Software online support provides an efficient way to access interactive technical support tools. As a valued support customer, you can benefit by using the support site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract.

To find more information about access levels and HP Passport, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Contents

1 Introduction

HP Select Identity (Select Identity) supports the ability to invoke calls to external systems. This guide provides technical information about the structure and architectural context of external calls. It also describes how to develop them, and provides code samples.

Use external calls to perform the following types of tasks:

- Generate a user ID or password. See Attribute Value Generation External Call Type on page 13.
- Provide a list of possible attribute values. See Attribute Value Constraint External Call Type on page 14.
- Validate the value of an attribute. See Attribute Value Validation External Call Type on page 15.
- Verify the value of an attribute. See Attribute Value Verification External Call Type on page 16.
- Query an external system for a list of approvers. See Approver Selection External Call Type on page 16.
- Perform a workflow task. See Workflow Action External Call Type on page 17.
- Retrieve a certificate from an external system. See Certification Management Function External Call Type on page 19.
- Convert an extended SPML file containing account reconciliation data into standard SPML, or customize an SPML request. See SPML Request Filter External Call Type on page 20.
- Group request targets to consolidate the number of requests generated during the Select Identity Multi-User Add Services/Attribute Values and Multi-User Delete Attribute Values operations. See External Request Process and Classes on page 10.

Java Classes and Interfaces

Creating an external call entails coding one or more Java classes. You must be familiar with the Java Developer Kit (JDK), version 1.4 or later. For information about the J2EE APIs, refer to **http://java.sun.com/j2se/1.4.2/docs/api/index.html**.

You must code the classes called by external calls using the External Call and Workflow APIs. These APIs define Java-based interfaces for creating external callouts. The Select Identity-facing portion of the interface must be written in Java.

JavaDoc for External Calls

JavaDoc is provided for the External Call API and Workflow API in the /`docs/api_help/external_calls/Javadoc` directory on the Select Identity product CD. Refer to this Web-based help for implementation and usage details.

External Call API Value Classes

The External Call API provides the following classes and interfaces, which are available for creating value external calls:

- **SIConstraintIntf**

  This interface must be implemented by classes that generate possible values for an attribute. For example, you would implement this interface in an external call that provides a list of department codes for presentation as options in a form, or to validate data. It is recommended that you use this interface, and not **TAValueConstraintIntf**, when developing new external calls.

- **TAValueConstraintIntf**

  This interface must be implemented by classes that provide a set of possible values for an attribute. The **SIConstraintIntf** interface is an extension of this interface.

- **TAValueConstraintIntf.TAValueConstraintBeanIntf**

  External calls that generate or provide a set of possible values for an attribute must return the values in a Java bean that implements this interface. For example, this applies to the department codes returned by the external call.

- **TAValueGenerationIntf**

  This interface must be implemented by external call classes that generate attribute values. For example, a class that implements this interface can generate a random password for a user.

- **TAValueValidationIntf**

  This interface must be implemented by external call classes that validate attribute values. For example, if a user enters a value into a form, the implementing class validates the value via this interface.

- **TAPolicyVerificationIntf**

  This interface must be implemented by external calls that validate the value of an attribute for a particular user. For example, you use a class that implements this interface to verify that a user's password stored outside of Select Identity is correct. Currently, only password attributes are verified in this way.

- **TAAttributeDefinitionException** and **TAAttributeValueValidationException**

  Exceptions defined for use by external calls.

- **ISpmlRequestFilter**

  This interface is implemented by the SPML Filter external call. It applies a filter to an incoming SPML request, and may convert the request to a new request as a result.

Workflow API Approver Selection and Workflow Classes

The Workflow API provides the following classes and interfaces for use in approver selection and workflow external calls.

- **WfSelectApproverIntf**

  This interface is the contract between Select Identity and an external entity to select approvers. External calls that implement this interface must return a collection of approvers. The external calls that implement this interface can be registered with Select Identity and used in approval stages to dynamically select approvers.

- **IWfClient**

  This interface invokes a workflow template, thereby creating a workflow instance. It can also resume an inactivated workflow instance, or terminate an instance.

- **IWfQuery**

  This interface retrieves runtime and configuration information about a workflow at the template, instance, block, or activity level.

- **IWfDataUpdate**

  Updates workflow variables.

- **StatelessServiceObjectFactory**

  Creates a component that implements the **IWfClient**, **IWfQuery**, and **IWfDataUpdate** interfaces, and forces these interfaces to behave like stateless Enterprise Java Beans (EJB) while hiding EJB-specific code and deployment information.

- **WfExternalCall**

  External calls invoked by workflow instances must implement this interface. This interface provides the contract between Select Identity and an external stage in a workflow. Select Identity can invoke classes that implement this interface to perform actions in a workflow.

- **WfExternalCallException**

  Defines exceptions used by the Workflow API.

- **WfExternalCallStatus**

  Returns the status of an approval activity to Select Identity. Along with the status of the stage, it can also return changes to the user profile, including attributes and entitlements.

Select Identity Request and Approval Classes

Use the following classes to explore Select Identity requests and approval frameworks:

- **AttributeRecord**

  Represents an attribute.

- **ChangeRecord**

  Represents a change in an attribute. Objects of this class can be used to communicate changes to Select Identity from external calls. For details about ChangeRecord, refer to the JavaDoc provided in the `/docs/api_help/external_calls/Javadoc` directory on the Select Identity product CD.

- **Request**

Defines methods to retrieve and set all information related to an incoming request in the Select Identity system.

- **RequestJobItem**

  Defines the job that handles the request in the Select Identity system.

- **RequestTarget**

  Defines the target of a request.

- **RequestTargetParam**

  Defines the parameters of a request target.

- **RequestTargetParamValue**

  Gets parameter values.

- **TAFilter**

  A general class that stores filter criteria for a selection (search) procedure.

- **TARequestAction**

  Represents the action to take place on the target resource.

- **TARequestEvent**

  Represents the event for the request.

- **TARequestType**

  Represents the type of request.

External Request Process and Classes

Use the following classes to group request targets to consolidate the number of requests generated during the Select Identity Multi-User Add Services/Attribute Values and Multi-User Delete Attribute Values operations.

- **SIExternalRequestProcessIntf**

  This interface must be implemented by classes that group request targets and can be used to consolidate the number of requests generated during the Select Identity Bulk Add Attribute Values and Bulk Delete Attribute Values operations.

  For example, you could implement this interface in an external call that groups request targets by the manager attribute. Based on this example:

  1. Select Identity invokes the external call when an administrator submits requests for Multi-User Add/Delete Attribute Values operations.
  2. The external call groups the request targets by the manager attribute (or other identified business requirement) and returns a consolidated list of request target groups.
  3. Select Identity either:
     - Processes the list "as is" if the action and service context in the request target groups are the same.

       Or

     - Splits the list based on action and service context and then processes.

4 Select Identity submits a conslidated number of requests that are grouped by manager.

The external call that implements this interface should:

- Use the SIExternalRequestProcessResponse object to return the list to Select Identity.
- Not change any data in the request targets.
- Not add or remove any request targets.

You can register one implementation of this external call within Select Identity.

- **SIExternalRequestProcessCommand** — carries list of request targets and external call parameters.
- **SIExternalRequestProcessResponse** — returns the list of request target groups to Select Identity.

2 Default External Calls

Select Identity provides default external calls through which you interact with external systems for workflow steps and approver lookups. Each external call is defined within one of the following call types:

- **Attribute Value Generation** — generates a user name, user ID, password, and other attributes, such as the user's company or department. See Attribute Value Generation External Call Type on page 13.
- **Attribute Value Constraint** — provides a list of possible values for an attribute. See Attribute Value Constraint External Call Type on page 14.
- **Attribute Value Validation** — validates the value of an attribute. See Attribute Value Validation External Call Type on page 15.
- **Attribute Value Verification** — verifies the value of an attribute. See Attribute Value Verification External Call Type on page 16.
- **Approver Selection** — searches an external system for a list of users who can approve provisioning requests during a workflow. See Approver Selection External Call Type on page 16.
- **Workflow Action** — performs a task as part of a workflow, to integrate approval processes with external processes and systems. See Workflow Action External Call Type on page 17.
- **Certification Management** — enables you to retrieve a certificate from an external system. See Certification Management Function External Call Type on page 19.
- **SPML Filter** — filters SPML data files used for account reconciliation. See SPML Request Filter External Call Type on page 20.

Most external calls have predefined parameters that you can modify. The following sections list and describe the functionality of the external calls and their parameters, by call type.

Attribute Value Generation External Call Type

Attribute value generation external call types generate the name or ID of a user, the user's password, and any other attribute, such as the user's company or department. Attribute value generation external calls include the following:

- IDValueGeneration
- PasswordValueGeneration
- UserIDValueGeneration

IDValueGeneration External Call

The **IDValueGeneration** external call generates an attribute that is a unique number.

This external call has the following parameters:

Parameter Name	Parameter Value Description
Suffix	Use after the number.
Prefix	Use before the number.

PasswordValueGeneration External Call

The **PasswordValueGeneration** external call generates a password that may contain both letters and numbers. The password must contain at least one number, and the letters must be lowercase. The password value is constrained by the **minLength** and **maxLength** parameters. Special characters ("/", "+", "-") are not permitted.

Parameter Name	Parameter Value Description
minLength	The minimum password length.
maxLength	The maximum password length .

UserIDValueGeneration External Call

The **UserIDValueGeneration** external call generates a user ID based on another attribute.

Parameter Name	Parameter Value Description
MaxRetryAttempts	Maximum number of tries to create a unique ID.
Length	The number of characters (alphanumeric or special) in the generated ID.
AttributeName	Attribute name from which the user ID is generated (such as from email).

Attribute Value Constraint External Call Type

Attribute value constraint external call types provide a list of possible values for an attribute. Attribute value constraint external call types include the following:

- **Search Connector**
- **Search Table**

Search Connector External Call

The **Search Connector** external call constrains attributes based on the Select Identity resource name specified.

Parameter Name	Parameter Value Description
resource_name	Select Identity resource name.

Search Table External Call

The **Search Table** external call constrains attributes based on the specified **query** and **valuefield** parameters. The query is executed using the specified **poolname** parameter value.

Parameter Name	Parameter Value Description
query	Select query with two result columns.
valuefield	The name of the column used to derive the constraint value. Other result columns are used as display.
poolname	JNDI name for the data source and poolname for which the query is to be executed.

Attribute Value Validation External Call Type

Attribute value validation external call types validate the values of attributes against specified validation criteria. Attribute value validation external call types include the following:

- **IsAlphaNumeric**
- **ManageExpireValidation**
- **PasswordValidation**
- **AttributeValueValidation**

IsAlphaNumeric External Call

The **IsAlphaNumeric** external call validates if the attribute value is alphanumeric. No parameters are editable with this external call.

ManageExpireValidation External Call

The **ManageExpireValidation** external call validates the value of the **ExpirationDate** attribute, which must be more than 30 days from the current date. An error message results if the value of the **ExpirationDate** attribute is less than 30 days. No parameters are editable with this external call.

PasswordValidation External Call

The **PasswordValidation** external call validates that a password contains the minimum number of each type of characters specified.

Parameter Name	Parameter Value Description
Special Characters	Number of required non-alphanumeric or special characters.
List of Special Characters	Comma-delimited list of valid special characters.
Lower Case Letters	Number of required lowercase letters.
Upper Case Letters	Number of required uppercase letters.
Numerics	Number of required numeric values.
Letters	Number of required letters.

This external call is also used with password policy settings.

Attribute Value Verification External Call Type

The **AttributeValueVerification** external call type verifies the value of an attribute. No parameters are editable with this external call.

Approver Selection External Call Type

Approver selection external call types search an external system for a list of users who can approve provisioning requests during a workflow. The approver selection external call type includes **WFGetApproverSampleExtCall**.

WFGetApproverSampleExtCall External Call

The **WFGetApproverSampleExtCall** external call is a sample external call that specifies a list of users to use for approvals.

Parameter Name	Parameter Value Description
SampleApprovers	Comma-delimited list of Select Identity user IDs to use for approvals.

Workflow Action External Call Type

Workflow action external call types perform a task as part of a workflow, to integrate approval processes with external processes and systems. Workflow action external calls include the following:

- **LoadUserServices**
- **UserEnableDisableWFExtCall**
- **WorkflowCertificateRequest**
- **ExclusionRuleCall**

LoadUserServices External Call

The **LoadUserServices** external call adds services to a user based on a change of context. For an example of how to use this external call, see the topic titled *Scenario: Adding Services to a User* in the *HP Select Identity Workflow Studio Online Help*.

Parameter Name	Parameter Value Description
ServicesRule	Specifies the rule name.

UserEnableDisableWFExtCall External Call

The **UserEnableDisableWFExtCall** external call enables or disables a user account based on the value stored in a specified attribute.

Parameter Name	Parameter Value Description
AttributeName	Attribute name for which the value is checked.
EnableValue	If the value of the user attribute in **AttributeName** matches this parameter, the external call enables the user when the user is disabled.
DisableValue	If the value of the user attribute in **AttributeName** matches this parameter, the external call disables the user when the user is enabled.
UserName	Administrator with authority to modify users who use this external call.
Password	Administrator's password.
url	Web services URL.

WorkflowCertificateRequest External Call

The **WorkflowCertificateRequest** manages certificates. For information on using this external call, see the certificate management topics in the *HP Select Identity Workflow Studio Online Help*.

Parameter Name	Parameter Value Description
DN_FieldName	Attribute name storing the user's distinguished name (DN) from the certificate.
CertificateFieldName	Challenge password assigned at the time of user registration.

Parameter Name	Parameter Value Description
EmailTemplateName	Default email template from Select Identity, to send email to the user.
CertificateProviderName	Certificate provider name. In the case of Verisign, the name must be “Verisign.” In all other cases, the administrator can assign the name.
ExternalCallName	Name of the CA-specific Java class implementing validation and generation functions for the certificate.

ExclusionRuleCall External Call

The **ExclusionRuleCall** external call prevents users from receiving services when the users have existing or pending conflicting services or entitlements.

Parameter Name	Parameter Value Description
RuleName **UniqueServicePrefix**	Parameter specifying which rule to run.
WFVariableName **$RuleExclusionMsg**	Workflow variable specifying the result message.

Certification Management Function External Call Type

Certification management function external call types implement validation and generation functions for the certificate. Certification management function external call types include only **VerisignCertImpl**. For detailed information about Verisign certificate management, see the *HP Select Identity Workflow Studio Online Help*.

VerisignCertImpl External Call

The **VerisignCertImpl** external call is called by the **WorkFlowCertificateRequest** external call, which validates certificate requests. This external call has no editable paramenters. For more information, see the *HP Select Identity Workflow Studio Online Help*.

SPML Request Filter External Call Type

The **SPML Request Filter** external call type is invoked during reconciliation, to convert and filter an incoming SPML request. It converts an extended SPML request into a standard SPML request, or customizes an SPML request.

The filter class should implement the interface **ISpmlRequestFilter**. This interface class and its associated model and exception classes are packaged into `clientintf.jar`, which is distributed for external call implementation.

The SPML Request Filter external call type is not invoked for SMPL 2.0 requests since web services reconciliation requests don't support SPML 2.0.

Using the Extended SPML Request Filter

This section provides information about how to use the **ExtendedSpmlRequestFilter** External Call in a Select Identity system.

Configuration

Select Identity should provide the user interface mechanism to configure the **ExtendedSpmlRequestFilter** external call for a given resource. The selection should be similar to other types of external configuration. If the filter is not configured, incoming SPML requests are not processed.

Invocation

When an SPML request is handed over to the Select Identity reconciliation module (via file upload, Web service, or direct polling), the filter class is loaded from the resource configuration. The external call parameters defined in the configuration are also loaded.

Extended Request

The reconciliation module adds two extra parameters to the additional arguments map for the extended request:

- **Key: "UserName" —** The attribute name for the Select Identity user ID.
  - **Value:** The name of resource attribute name that mapped to this attribute.
- **Key: "Password" —** The attribute name for the Select Identity default password
  - **Value:** The name of resource attribute name that mapped to this attribute.

After filtering, Select Identity expects that the extended request is converted to another request, typically an SPML `<modifyRequest>`. The reconciliation modify process should follow after filtering.

Add/Modify/Delete request

The reconciliation module does not add parameters to SPML add, modify, or delete requests.

Sample Filter Implementation

The **ExtendedSpmlRequestFilter** external call handles the following extended requests:

- **Reset and Change Password**

  Reset and change password requests are converted to modify requests with the password as the modification. If key fields are not provided in the original request, the mapped Select Identity user ID attribute is inserted as the key field.

  The following extended request attributes are converted to a modification on the mapped Select Identity default password attribute:

  - `urn:trulogica:concero:2.0# rcPassword`
  - `urn:trulogica:concero:2.0# newPassword`
  - `urn:trulogica:concero:2.0# password`

  Other attributes are converted to modifications.

  The operation tags for all modifications are “REPLACE”.

- **Enable and Disable**

  Enable and disable requests are converted to modify requests

  If key fields are not provided in the original request, the mapped Select Identity user ID attribute is inserted as the key field.

  If the enable/disable requests do not contain any attributes that actually identify what is changed on the resource to cause connector agents to send enable/disable, then a modification with the name `“UserEnabled”, Value “1/0”, Operation=REPLACE` is inserted the modification list. The resource mapping file *must* contain an artificial field named **UserEnabled**. The corresponding Select Identity attribute should also be created. This artificial resource attribute should only allow sync-in but not sync-out.

3 Creating an External Call

The sections in this chapter describe how to implement an external call of each type. Refer to the JavaDoc provided in the `/docs/api_help/external_calls/Javadoc` directory on the Select Identity product CD for details about specific APIs.

Coding Attribute Value External Calls

To create an external call, follow these guidelines:

- **Attribute Value Generation** external calls must implement the **TAValueGenerationIntf** interface. See Attribute Value Generation External Call on page 35 for code examples.
- **Attribute Value Constraint** external calls must implement the **TAValueConstraintIntf** and **SIConstraintIntf** interfaces. See Attribute Value Constraint External Call on page 36 for code examples.
- **Attribute Value Validation** external calls must implement the **TAValueValidationIntf** interface. See Attribute Value Validation External Call on page 47 on for code examples.
- **Attribute Value Verification** external calls must implement the **TAPolicyVerificationIntf** interface.

Retrieving Parameters, Attributes, and Data

The following guidelines describe how to obtain input parameters, Select Identity attributes, and the data that needs to be returned:

- To retrieve an external call parameter defined on the Select Identity **External Calls** page, use the `containsKey` and `get` methods. The `containsKey` method verifies the existence of the specified parameter, and the `get` method retrieves the parameter. For example:

  ```
  if (attrs.containsKey("parametername"))
     String parameter = (String)
  attrs.get("parametername")
  ```

- To retrieve a Select Identity attribute, use one of the methods provided by the `RequestTarget` class, such as `getSingleRequestParamStr`. This example retrieves a single value attribute:

  ```
  String attributeValue=(String)requestTarget.
  getSingleRequestParamStr("attributename");
  ```

  To retrieve the values of a multivalue attribute, use the `getRequestParam` method. For example:

  ```
  Set attributeValues=(Set)requestTarget.
  getRequestParam("attributename").getRequestTArgetParamValue()
  ```

- Throw `TAAttributeDefinitionException` if the external call is unsuccessful.

- If you write an external call function that must access data in the Select Identity database, the function can acccess the database using the JNDI name specified by the `truaccess.dataSource` property in the `TruAccess.properties` file. Or, the function can access the Select Identity database in other ways depending on how you construct the function.

  For more information about the `TruAccess.properties` file, see the *HP Select Identity Installation Guide*.

- Use any logging mechanism to log messages and errors. The examples shown in this guide use an internal logging mechanism that is not available externally.

- Select Identity passes the following parameters for **ValueConstraint**:

  - **OVSIServiceName**
  - **OVSIServiceID**
  - **OVSIContextAttrName**
  - **OVSIContextAttrId**
  - **OVSIContextAttrValue**
  - **OVSIContextName**
  - **OVSIServiceRoleName**

  The following sample shows how to use the SIConstraint external call to get service and context information. The service name, context name, context values, and other data are passed to the SIConstraint external call.

```
public List getValueConstraint( String attribName,
                                 TAFilter filter,
                                 Map args )
                                 throws TAAttributeDefinitionException {
// name of the service
String serviceName = ( String )args.get(
                                      SIConstraintIntf.OVSI_SERVICE_NAME);
// identifier of the service
String serviceIdStr = ( String ) args.get(
                                      SIConstraintIntf.OVSI_SERVICE_ID);
       // name of the context attribute
String contextAttrName = ( String ) args.get(
                                      SIConstraintIntf.OVSI_CTX_ATTR_NAME);
// Identifier of the context attribute
String contextAttrIdStr = ( String ) args.get(
                                      SIConstraintIntf.OVSI_CTX_ATTR_ID);
// Value of context attribute that user selects
//in registration form
String contextAttrValue = ( String ) args.get(
                                      SIConstraintIntf.OVSI_CTX_ATTR_VALUE);
// name of context name as defined in Service Context
                                      String contextName = ( String ) args.get(
SIConstraintIntf.OVSI_CTX_NAME);
// name of the service role as defined in Service Role
String serviceRoleName = ( String ) args.get(
             SIConstraintIntf.OVSI_SR_NAME);
try{
// write code to use these values and return List of object of
// type TAValueConstraintIntf.TAValueConstraintBeanIntf
}
catch (Exception e) {
       // throws TAAttributeDefinitionException exception in
       // of any error
throw new TAAttributeDefinitionException(e.getMessage());
}



}
```

- Select Identity passes the following parameters for **ValueValidation**:
  - **UserName**
  - **BizKey** (of the service)
  - **GUID**
  - **HPSIRequestTarget**

    When adding a new user, `RequestTarget` retains all attribute values . For other request types, `RequestTarget` *only* retains modified values. An attribute value validation external call may therefore be required to query the Select Identity database for other needed values.

    An attribute value validation external call can retrieve `RequestTarget` from the maps argument by using the `PARMAM_REQUEST_TARGET` constant.

    ```
    RequestTarget  reqTarget = (RequestTarget)
                       args.get("PARAM_REQUEST_TARGET");
    ```

    For example, assume you want to validate the value of the `Email` attribute against the following format:

    ```
    <FirstName>.<LastName>@<company>.com
    ```

    Using the `PARMAM_REQUEST_TARGET` constant, an attribute value validation external call can get the values of the `FirstName`, `LastName`, and `Company` attributes from the `RequestTarget` and validate them against the value of the `Email` attribute from the `RequestTarget`.
- Validation errors returned from external calls are displayed on the Select Identity user interface.

Coding Approver Selection External Calls

For approver selection external calls, the APIs support synchronous communication. The external system must complete its processing and provide status information as part of the call. The call is required to return status indicating how Select Identity will proceed with the workflow.

External Call Variables

The following variables are available for use in your approver selection external call:

Variable Name	Description
_activityId	The activity ID string for the activity currently in execution. This variable is maintained by the engine and cannot be updated in the template. A string value is assigned to this variable.
_instActivityId	An internal variable representing the instance activity ID for the current activity in execution. This variable is maintained by the engine and cannot be updated in template. The variable is assigned an integer value.
_joinCommand	Passed into the workflow by an external application invoked as an action in a workflow template activity. This variable is not persistent. Possible values are as follows: • `exit` — exits the block unconditionally. • `exitAll` — exits the current block and all parent blocks unconditionally. • `reset` — resets the value of the **joinCount** property set in the block. For example, the application could pass this variable to the workflow to instruct it to exit a block unconditionally (even if, for instance, the `_joinCount` value is not met).
_pushVar	Passed into the workflow by an external application invoked as an action in a workflow template activity. The object is then added to workflow's internal `pushList` block variable. The pushed objects in the list can be displayed later in tabular format in a report. You can specify any Java object as the value of this variable.
$_blockId	The block ID for the current block. You can assign any string value to this variable.
$_instId	The workflow instance ID. This variable is maintained by the engine and cannot be updated. An integer value is assigned to this variable.

Implementation

The following provides general information about how to implement an approver selection external call, how to obtain to input parameters and Select Identity attributes, and what data needs to be returned.

- Approver selection external calls must implement the **WfSelectApproverIntf** interface.
- The main method of the external call must be called `getApprover`. Here is the call signature of the `getApprover` method:

```
public Collection getApprover(RequestTarget reqTarget,
HashMap attrs)
throws WfExternalCallException;
```

- The Workflow external call must return a collection of Select Identity user IDs. In addition, the external call must return data that is valid in Select Identity.

  

  If the external system cannot return a valid Select Identity user ID, the workflow process waits for a valid ID. You must terminate the request and retry.

- To retrieve an external call parameter defined on the Select Identity **External Calls** page, use `containsKey` and `get` methods. The `containsKey` method verifies the existence of the specified parameter, and the `get` method retrieves the parameter. For example:

```
if (attrs.containsKey("parametername"))
   String parameter = (String)
attrs.get("parametername")
```

Standard Parameters

Select Identity defines the following standard parameters in the map:

- `WfExternalCall.WF_PARAM_SERVICENAME` — for the service name. This is a string object.
- `WfExternalCall.WF_PARAM_ADMINUSERID` — for the requestor's user name. This parameter is empty for a self-registration or system-generated request. This is a string object.
- `WfExternalCall.WF_PARAM_REQUESTID` — for the request identifier. This is an integer object.
- `WfExternalCall.WF_PARAM_WORKFLOWINSTID` — for the workflow instance ID. This is an nteger object.

Retrieving Attributes and Variables

To retrieve a Select Identity attribute, use one of the methods provided by the **RequestTarget** class, such as `getSingleRequestParamStr`. For example, you may need to populate the values for each user attribute defined by a map in an external call that is called by a workflow template.

The following examples illuatrate various types of attribute and variable retrieval:

- Here is an example that retrieves a single-value attribute:

```
String attributeValue=(String)requestTarget.
getSingleRequestParamStr("attributename");
```

- To retrieve the values of a multi-value attribute, use the following method:

```
Set attributeValues=(Set)requestTarget.
getRequestParam("attributename").getRequestTargetParamValue()
```

  For a more extensive example, see Retrieving Request Object Data to Set Workflow Variables on page 54.

- To retrieve workflow variables, use the **IWfQuery** interface as in the following example:

```
IWfQuery query =(IWfQuery)StatelessServiceObjectFactory.
create(IWfQuery.class);

int instanceId = query.
getInstanceInfoByInstActivityId(instActivityId).getInstId();
Map varMap = query.getCurrentVariableMap(instanceId);
String comment = (String) varMap.get("$ApproverComments");
```

- To set and update a workflow variable, use the methods provided by the **IWfDataUpdate** interface, as follows:

```
IWfDataUpdate du =(IWfDataUpdate)StatelessServiceObjectFactory.
     create(IWfDataUpdate.class);
du.setWorkflowVar(instId, "workflowvariablename", "value");
```

- A workflow variable can be any Java object. However, if it is a persistent variable with its name preceded by `$`, you may either use HashMap to contain object attributes, or use the Java object marked as Serializable. If the persistent variable is a Java object, add following line to the Java class:

```
static final long serialVersionUID = uid;
```

  where `uid` can be `0` for the newly created class or found by running JDK's server command.

- You can use any logging mechanism to log messages and errors. The examples in this guide use an internal logging mechanism that is not available externally.

Approver Selection External Call Code Example

This code example illustrates how to select a manager as an approver. Request or user attributes must contain **ManagerId** attribute. This attribute name can be passed through an external call parameter.

```
public Collection getApprover(RequestTarget reqTarget,
                              HashMap attrs)
              throws WfExternalCallException{

       ArrayList ret = new ArrayList();
       try{
              String query = (String) params.get(PARAM_QUERY);
              String jndiPool = (String) params.get(PARAM_POOL);
              String mgrAttrName = (String) attrs.get("MANAGER_ATTR");

              String managerUserId = reqTarget.getParamValueString(
                     mgrAttrName);

              if (managerUserId == null){
              String userName = reqTarget.getKeyFieldValue();
              if (userName == null){
                     WfExternalCallException exp =
                            new WfExternalCallException(
                            "UserName can not be null");
                     throw exp;
              }

              String query2getMgrIdFromDB = "select stringValue from"
              + " TAUser u, TAAttribute a where u.userId = "
              + " a.identObjId and u.conceroUserId = "
              + "'" + username + "'" + " and a.name = '"
              + managerAttributeName + "'";

              managerUserId = getAttributetValueFromDB (
                         query2getMgrIdFromDB, jndiPool );

              if (managerUserId == null){
                     WfExternalCallException exp = new
                            WfExternalCallException(
                     "Unable to get manager Id from user profile");
                     throw exp;
              }

              }// managerUserId == null

              if (managerUserId == null){
                     WfExternalCallException exp = new
```

```
                        WfExternalCallException(
                        "Unable to get approver Id for this request");
                throw exp;
            }
            ret.add(managerUserId);
            return ret;
        }
        catch (Exception e){
            WfExternalCallException exp = new
                        WfExternalCallException(
                        e.getMessage());
                throw exp;
        }

        return ret;
    }
```

Coding Workflow Action External Calls

For workflow action external calls, the APIs support synchronous communication. Select Identity requires the external system to complete its processing and provide status information as part of the callout. The callout is required to return status indicating how Select Identity proceeds with the workflow.

Implementation

The following provides information about how to implement a workflow action external call, how to obtain to input parameters and Select Identity attributes, and what data needs to be returned:

- Workflow external calls must implement the **WfExternalCall** interface.
- The main method of the external call must be called `process`. This method expects four input parameters.

  The following is an example of the call signature for the `process` method:

  ```
  WfExternalCallStatus process(String stageId,
  RequestTarget requestTarget,
  AttributeRecord [] availGrp,
  AttributeRecord [] availRole,
  AttributeRecord [] availEntilements,
  Map map) throws WfExternalCallException
  ```

- The workflow action external call must return `WfExternalCallStatus`. In addition, the external call must return data that is valid in Select Identity.
- To retrieve an external call parameter defined on the Select Identity **External Calls** page, use the `containsKey` and `get` methods. The `containsKey` method verifies the existence of the specified parameter, and the `get` method retrieves the parameter, as in the following example:

  ```
  if (attrs.containsKey("parametername"))
     String parameter = (String)
  attrs.get("parametername")
  ```

- To log messages and errors, you can use any logging mechanism. The examples shown in this guide, however, use an internal logging mechanism that is not available externally.

Standard Parameters

Select Identity defines the following standard parameters in the map:

- `WfExternalCall.WF_PARAM_SERVICENAME` — for the service name. This is a string object.
- `WfExternalCall.WF_PARAM_ADMINUSERID` — for the requestor's user name. This is a string object.
- `WfExternalCall.WF_PARAM_REQUESTID` — for the request identifier. This is an integer object.
- `WfExternalCall.WF_PARAM_WORKFLOWINSTID` — for the workflow instance ID. This is an integer object.

Retrieving Attributes and Variables

To retrieve a Select Identity attribute, use one of the methods provided by the **RequestTarget** class, such as `getSingleRequestParamStr`.

- The following example retrieves a single-value attribute:

```
String attributeValue=(String)requestTarget.
getSingleRequestParamStr("attributename");
```

- To retrieve the values of a multi-value attribute, use the `getRequestParam` method. For example:

```
Set attributeValues=(Set)requestTarget.
getRequestParam("attributename").getRequestTargetParamValue()
```

- To retrieve workflow variables, use the **IWfQuery** interface, as in the following example:

```
IWfQuery query =(IWfQuery)StatelessServiceObjectFactory.
create(IWfQuery.class);

int instanceId = query.
getInstanceInfoByInstActivityId(instActivityId).getInstId();
Map varMap = query.getCurrentVariableMap(instanceId);
String comment = (String) varMap.get("$ApproverComments");
```

- To set and update a workflow variable, use the methods provided by the **IWfDataUpdate** interface, as in the following example:

```
IWfDataUpdate du =(IWfDataUpdate)StatelessServiceObjectFactory.
     create(IWfDataUpdate.class);
du.setWorkflowVar(instId, "workflowvariablename", "value");
```

- Update a Select Identity attribute using ChangeRecord as follows:

```
WfExternalCallStatus ecs = new WfExternalCallStatus(stageId);
ChangeRecord changeRecord = new ChangeRecord();

// Update the attribute in the change record
changeRecord.setName("attributename");
changeRecord.addValue("newattributevalue");

ecs.addAttributeChange(changeRecord);
```

For details about ChangeRecord, refer to the JavaDoc provided in the `/docs/api_help/external_calls/Javadoc` directory on the Select Identity product CD.

Updating Workflow Variables

Update workflow variables that need to be persisted using `setStatus()` provided by the **WfExternalCallStatus** class. All variable names starting with `$` are assumed to be workflow variables and are persisted when the call returns.

- The default ChangeRecord operation is ADD. For details about ChangeRecord, refer to the JavaDoc provided in the `/docs/api_help/external_calls/Javadoc` directory on the Select Identity product CD.
- To set the return status of a workflow external call, use the **WfExternalCallStatus** class and methods, as follows:

  ```
  ecs = new WfExternalCallStatus(stageId);
  ```

  If the call returns successfully, use the following:

  ```
  ecs.setStatus(WfExternalCallStatus.STATUS_APPROVED);
  ```

  If the call is unsuccessful, use the following:

  ```
  ecs.setStatus(WfExternalCallStatus.STATUS_REJECT_TERMINATE);
  ```

External Call Variables

The following variables are available for use in your external call:

Variable Name	Description
_activityId	The activity ID string for the activity currently in execution. This variable is maintained by the engine and cannot be updated in the template. A string value is assigned to this variable.
_instActivityId	An internal variable representing the instance activity ID for the current activity in execution. This variable is maintained by the engine and cannot be updated in template. The variable is assigned an integer value.
_joinCommand	Passed into the workflow by an external application invoked as an action in a workflow template activity. This variable is not persistent. Possible values are as follows: • `exit` — exits the block unconditionally. • `exitAll` — exits the current block and all parent blocks unconditionally. • `reset` — resets the value of the **joinCount** property set in the block. For example, the application could pass this variable to the workflow to instruct it to exit a block unconditionally (even if, for instance, the `_joinCount` value is not met).

Variable Name	Description
_pushVar	Passed into the workflow by an external application invoked as an action in a workflow template activity. The object is then added to workflow's internal `pushList` block variable. The pushed objects in the list can be displayed later in tabular format in a report. You can specify any Java object as the value of this variable.
$_blockId	The block ID for the current block. You can assign any string value to this variable.
$_instId	The workflow instance ID. This variable is maintained by the engine and cannot be updated. An integer value is assigned to this variable.

Workflow Action External Call Code Examples

This example illustrates how to get the JDBC connection in an external call. The JDBC connection pool name can be passed through external call parameters.

```
protected Connection getConnection(String connectionPoolName)
        throws Exception {
        InitialContext ctx = null;
        try {
                ctx = new InitialContext();
                DataSource ds =(DataSource)ctx.lookup(connectionPoolName);
                return ds.getConnection();
        } finally {
                try {
                        if (null != ctx) {
                                ctx.close();
                        }
                        ctx = null;
                        }
                        catch (Throwable t) {
                                //log.warn("Error closing context", t);
                        }
        }//end finally
        //end getConnection
}
```

This example illustrates how to get an attribute value from a request and set it as a workflow variable. The attribute name and workflow variable name must begin with "__" and can be passed through external call parameters.

```
public WfExternalCallStatus process(String stageId,
                RequestTarget reqTarget,
                AttributeRecord[] availableGroups,
                AttributeRecord[] availableRoles,
                AttributeRecord[] availableEntitlements, Map params)
                throws WfExternalCallException {

        try {
                String attrName = (String) params.get(ATTR_NAME);
                // workflow variable name starts with '__' if it
                // is set using ChangeRecord
                String wfVarName = (String) params.get(WF_VAR_NAME);

                //
                String attrValue  =
                        reqTarget.getParamValueString(attrName);

                //
```

```
                ChangeRecord chRec = new ChangeRecord();
                chRec.setName(wfVarName);
                chRec.addValue(attrValue);

                WfExternalCallStatus status = new
                        WfExternalCallStatus(stageId);
                status.addAttributeChange(chRec);
                status.setStatus(WfExternalCallStatus.STATUS_APPROVED);
                return status;

        } catch (Throwable t) {
                WfExternalCallException exp = new
                        WfExternalCallException(
                                "Unable to lookup the Attribute value ");
                throw exp;
        }
}
```

Coding External Request Process External Calls

This type of external call must implement the **SIExternalRequestProcessIntf** interface.

Retrieving Parameters and Request Targets

The following guidelines describe how to obtain input parameters, Select Identity attributes, and the data that needs to be returned:

- To retrieve an external call parameter defined on the Select Identity **External Calls** page, use the `SIExternalRequestProcessCommand` and `getParams()` methods. For example:

  ```
  final SIExternalRequestProcessCommand reqCmd =
     (SIExternalRequestProcessCommand)command;
  Map params = reqCmd.getParams();
  String keyAttrName = (String)params.get("MGR_ATTR");
  String keyAttrValues = (String)params.get("MGR_NAMES");
  ```

- To retrieve a Select Identity request target, use the `getReqTargetList()` method. For example:

  ```
  Collection reqTargetList = reqCmd.getReqTargetList();
  ```

- To build the response to return, use the `addRequestTargetList()` method. For example:

  ```
  SIExternalRequestProcessResponse response =
     (new SIExternalRequestProcessResponse();
  response.addRequestTargetList(resultList1);
  response.addRequestTargetList(resultList2);
  response.addRequestTargetList(resultList3);
  return response;
  ```

- Throw `CommandException` if the external call is unsuccessful.

Sample Code Using the SIExternalRequestProcess External Call

The following sample shows how to use the **SIExternalRequestProcess** external call to group request targets by manager attribute.

```
public Response execute(Command command) throws UnsupportedOperationException,
                    CommandException {
       final int commandCode = command.getCommand();
       if (commandCode == SIExternalRequestProcessIntf.REQUEST_SPLIT_COMMAND){
              try {
                    final SIExternalRequestProcessCommand reqCmd =
                           (SIExternalRequestProcessCommand)command;
                    Map params = reqCmd.getParams();
                    // use this param to retrieve external call setting in SI UI
                    String keyAttrName = (String)params.get("MGR_ATTR");
                    String keyAttrValues = (String)params.get("MGR_NAMES");
                    // assume three manager names in keyAttrValues, split the
                    request depends on three manager
                    // manager1, manager2, manager3
                    Collection reqTargetList = reqCmd.getReqTargetList();
                    // getListByManager() method goes through list
                    Collection resultList1 = getListByManager(reqTargetList,
                           keyAttrName, manager1);
                    Collection resultList2 = getListByManager(reqTargetList,
                           keyAttrName, manager2);
                    Collection resultList3 = getListByManager(reqTargetList,
                           keyAttrName, manager3);

                    SIExternalRequestProcessResponse response =
                           new SIExternalRequestProcessResponse();
                    response.addRequestTargetList(resultList1);
                    response.addRequestTargetList(resultList2);
                    response.addRequestTargetList(resultList3);
                    return response;
              }
              catch (Exception e) {
                    throw new CommandException(e.getMessage(), e);
              }
       }
       throw new UnsupportedOperationException("Upsupported Command.");
       //end execute
}
```

4 Examples

This chapter provides examples for each type of external call. Refer to the Javadoc in the `/docs/api_help/external_calls` and `workflow` directories on the Select Identity product CD for information about the APIs.

These examples are included in the Select Identity EAR file deployed during installation. If you wish to modify and register one of the example calls, you must change the class name.

Attribute Value Generation External Call

This class generates the password from a Social Security Number or ID. The configuration arguments to the function are the `SSN_FIELD` and `PERSONNUMBER_FIELD` attributes. The supporting class, `UserNameValueBean.java`, follows this example on page 36.

```
package com.trulogica.truaccess.externalcall.generatefunction;

import java.util.StringTokenizer;
import java.util.Random;
import java.util.Map;
import com.trulogica.truaccess.request.model.RequestTarget;
import com.trulogica.truaccess.attribute.TAValueGenerationIntf;
import com.trulogica.truaccess.attribute.exception.
TAAttributeDefinitionException;
import com.trulogica.truaccess.util.logging.misc.Logger;
import com.trulogica.truaccess.util.logging.misc.Level;
import com.trulogica.truaccess.externalcall.generatefunction.
UserNameValueBean;
/**
 * Generates a String ID with the format Prefix + ID + Suffix
 * Prefix and Suffix are specified as parameters to the call
 */
public class SimpleValueGenerator implements TAValueGenerationIntf
{
     public Object generateValue(String attribName,
                                 RequestTarget reqTarget, Map args)
                                 throws TAAttributeDefinitionException
     {

          try {

               if (mLogger.isLoggable(Level.FINEST))
                    mLogger.finest("Generating the value of :"+attribName);
               if (mLogger.isLoggable(Level.FINEST))
                    mLogger.finest("RequestTarget:"+reqTarget.toString());

               String prefix = (String)args.get(PARAM_PREFIX);
            String suffix = (String)args.get(PARAM_SUFFIX);

            StringBuffer sb = new StringBuffer();

            sb.append((null==prefix)?"":prefix).append((new

           Random()).nextInt()).append((null==suffix)?"":suffix);
```

```
            if (mLogger.isLoggable(Level.FINEST))
                mLogger.finest("Returning a value of : " + sb.toString() +
                                        " : for attribute" + attribName);
            return new UserNameValueBean(sb.toString());

          } catch (Throwable t) {
              if (mLogger.isLoggable(Level.WARNING))
                   mLogger.log(Level.WARNING,"Unable to generate attribute
                   value for attribute:"+attribName,t);
              TAAttributeDefinitionException exp =  new
                   TAAttributeDefinitionException("Unable to generate
                   attribute value for
                   attribute:"+attribName+t.getMessage());
              exp.setCausedBy(t);
              throw exp;
          }
    }

    private static final Logger mLogger =
          Logger.getLogger(SimpleValueGenerator.class.getName());
    private static final String PARAM_PREFIX="prefix";
    private static final String PARAM_SUFFIX="suffix";
}
```

The value generation function above calls the class below:

```
package com.trulogica.truaccess.externalcall.generatefunction;
import com.trulogica.truaccess.attribute.TAValueConstraintIntf.
TAValueConstraintBeanIntf;
public class UserNameValueBean implements TAValueConstraintBeanIntf

{
     String value = null;
     public UserNameValueBean(String _value)
     {
          value = _value;
     }
     public String getName() {
          return null;
     }

     public Object getValue() {
          return value;
     }
}
```

Attribute Value Constraint External Call

The following examples implement a simple table-based lookup of possible constraint values. The examples use external arguments —poolname and SQL string— to query the database tables. The supporting class, `SearchResult.java`, follows the main classes.

Implementing the TAValueConstraintIntf Interface

```
package com.trulogica.truaccess.externalcall.constraintfunction;

import java.util.*;
import com.trulogica.truaccess.util.logging.misc.Logger;
```

```
import com.trulogica.truaccess.util.logging.misc.Level;

import com.trulogica.truaccess.base.TAFilter;
import com.trulogica.truaccess.attribute.TAValueConstraintIntf;
import com.trulogica.truaccess.attribute.exception.
TAAttributeDefinitionException;

import com.trulogica.truaccess.externalcall.constraintfunction.
SearchResult;

import com.trulogica.truaccess.util.Tools;
import com.trulogica.truaccess.util.DBTool;
import javax.naming.InitialContext;
import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Types;

/**
 * This class implements simple table-based lookups of possible constraint
 * values. It uses the external arguments poolname and sql string to query
 * the database tables.
 */
public class SearchTable implements TAValueConstraintIntf {

     private static final Logger mLogger =
      Logger.getLogger(SearchTable.class.getName());

     public static final String PARAM_POOLNAME = "poolname";
     public static final String PARAM_SQLSTRING = "query";
     public static final String PARAM_VALUEFIELD = "valuefield";

     private String poolname = null;
     private String query = null;
     private String valuefield = null;

     /**
      * Returns a List of all possible values.
      * Since this list can be very large it should be used sparingly.
      * @param attribName
      * @param args
      * @return
      * @throws TAAttributeDefinitionException
      * @see com.trulogica.truaccess.attribute.TAValueConstraintIntf
      */

     public List getValueConstraint(String attribName, Map args) throws
     TAAttributeDefinitionException
     {
          return getValueConstraint(attribName,null,args);
     }

     public List getValueConstraint(String attribName,TAFilter filter, Map
     args) throws TAAttributeDefinitionException
     {
          Connection connection = null;
          PreparedStatement pStmt = null;
          ResultSet rs = null;
          ArrayList ar = new ArrayList();
          try {
               poolname = (String)args.get(PARAM_POOLNAME);
               query = (String)args.get(PARAM_SQLSTRING);
               valuefield = (String)args.get(PARAM_VALUEFIELD);

               checkParam(poolname,PARAM_POOLNAME);
               checkParam(query,PARAM_SQLSTRING);
               checkParam(valuefield,PARAM_VALUEFIELD);

               if (mLogger.isLoggable(Level.FINEST)) mLogger.finest("Trying
               to locate values for attribute:"+attribName);
               String modFilter = "";
```

```
                    boolean hasParam = false;
                    if (null != filter) {

                         switch (filter.getOperation()) {
                              case TAFilter.EQUALITY:
                                   modFilter = "  "+valuefield+" = ?";
                                   break;
                              case TAFilter.BEGINS_WITH:
                                   modFilter = " "+valuefield+" LIKE ?";
                                   filter.setValue(filter.getValue()+"%");
                                   break;
                              case TAFilter.ENDS_WITH:
                                   modFilter = " "+valuefield+" LIKE ?";
                                   filter.setValue("%"+filter.getValue());
                                   break;
                              case TAFilter.CONTAINS:
                                   modFilter = " "+valuefield+" LIKE ?";
                                   filter.setValue("%"+filter.getValue()+"%");
                                   break;
                         }

                         String query2Check = query;
                         query2Check = query2Check.toUpperCase();

                         if (modFilter.length() > 0) {
                              if (query2Check.indexOf(" WHERE ") > 0) {
                                   query = query + " AND " + modFilter;

                              } else {
                                   query = query + " WHERE " + modFilter;
                              }
                              hasParam = true;
                         }
                    }
                    connection = this.getConnection();

                    pStmt = connection.prepareStatement(query);

                    if (mLogger.isLoggable(Level.FINEST))
                     mLogger.finest("query:("+query+"):modfilter:"+modFilter);
                    if (hasParam) {
                         pStmt.setString(1,filter.getValue());
                    }

                    rs = pStmt.executeQuery();
                    ResultSetMetaData rsMetaData = rs.getMetaData();
                    int nameColumn=0;

               //Select the first String field whose column name does not match the
               //value field

                    for (int i = 1; i <= rsMetaData.getColumnCount();i++)
                    {
                         if (((rsMetaData.getColumnType(i)==Types.VARCHAR)||
                                   (rsMetaData.getColumnType(i)==Types.CHAR))
                         &&(!rsMetaData.getColumnName(i).equalsIgnoreCase
                         valuefield))) {
                                   nameColumn=i;
                                   break;
                         }

                    }

                    while (rs.next())
                    {
                         SearchResult sr = new SearchResult();

                         sr.setValue(rs.getString(valuefield));

                         if (nameColumn>0)
                              sr.setName(rs.getString(nameColumn));
                         else
                              sr.setName((String) sr.getValue());
```

```
                    ar.add(sr);
                }

                return ar;
            } catch (Exception e) {
                if (mLogger.isLoggable(Level.WARNING))
                 mLogger.log(Level.WARNING,"Unable to obtain the values",e);
                TAAttributeDefinitionException exp =  new
                 TAAttributeDefinitionException();
                exp.setCausedBy(e);
                throw exp;
            } finally {
                DBTool.close(rs);
                DBTool.close(pStmt);
                DBTool.close(connection);
            }
        }

        private Connection getConnection() throws Exception
        {
            InitialContext ctx = null;

            try {
                ctx = new InitialContext();
                DataSource ds = (DataSource)ctx.lookup(poolname);
                return ds.getConnection();
            } finally {

                Tools.close(ctx);
            }

        }

        private void checkParam(String value,String name) throws Exception
        {
            if ((null == value)||(value.trim().length() == 0))
                throw new Exception("The value of "+name+" is needed");
        }
}
```

The following code is called by the class listed above.

```
package com.trulogica.truaccess.externalcall.constraintfunction;

import com.trulogica.truaccess.attribute.TAValueConstraintIntf;

public class SearchResult implements
TAValueConstraintIntf.TAValueConstraintBeanIntf
{
    private String name;
    private Object value;

    public SearchResult() {
    }

    public String getName() {
        return name;
    }

    public void setName(String _name) {
        name = _name;
    }

    public Object getValue() {
        return value;
    }
```

```
     public void setValue(Object _value)
     {
          value = _value;
     }
}
```

Implementing the SIConstraintIntf Interface

```
package com.trulogica.truaccess.externalcall.constraintfunction;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Types;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import com.hp.ovsi.attribute.SIConstraintIntf;
import com.hp.ovsi.attribute.api.GetConstraintsCommand;
import com.hp.ovsi.attribute.api.GetConstraintsResponse;
import com.hp.ovsi.base.Command;
import com.hp.ovsi.base.Response;
import com.trulogica.truaccess.attribute.exception.
     TAAttributeDefinitionException;
import com.trulogica.truaccess.base.TAFilter;
import com.trulogica.truaccess.base.TAQuery;
import com.trulogica.truaccess.base.exception.CommandException;
import com.trulogica.truaccess.util.DBTool;
import com.trulogica.truaccess.util.Tools;

public class SearchTable implements SIConstraintIntf //TAValueConstraintIntf
{

          private static final Log LOGGER =
               LogFactory.getLog(SearchTable.class );

    public static final String PARAM_POOLNAME = "poolname";
    public static final String PARAM_SQLSTRING = "query";
    public static final String PARAM_VALUEFIELD = "valuefield";

    private String poolname = null;
    private String query = null;
    private String valuefield = null;

    /**
     * Default NoArg Constructor
     */
    public SearchTable() {
        //end <init>
    }

    public SearchTable(final Map args) throws
                TAAttributeDefinitionException{
        setParams(args);
        //end <init>
    }

    /**
     * Returns a List of all possible values. Because this list can be
          * potentially very large, it should be used sparingly.
     *
     * @param attribName
     * @param args
     *
```

```
 * @return
 *
 * @throws TAAttributeDefinitionException
 * @see com.trulogica.truaccess.attribute.TAValueConstraintIntf
 */
public List getValueConstraint( String attribName, Map args ) throws
            TAAttributeDefinitionException {
    return getValueConstraint( attribName, null, args );
}

/**
 * @see com.trulogica.truaccess.attribute.TAValueConstraintIntf#
       * getValueConstraint(java.lang.String,
       com.trulogica.truaccess.base.* TAFilter, java.util.Map)
 */
public List getValueConstraint( String attribName, TAFilter filter,
            Map args ) throws TAAttributeDefinitionException {
    PreparedStatement pStmt = null;
    ResultSet rs = null;
    ArrayList ar = new ArrayList();
    //FIXME: Why store the params, when replaced after every call?
    setParams(args);

    if ( LOGGER.isTraceEnabled() ) {
        LOGGER.trace( "Trying to locate values for attribute:" +
                                  attribName );
    }
    String modFilter = "";
    boolean hasParam = false;
    if ( null != filter ) {

        switch ( filter.getOperation() ) {
        case TAFilter.EQUALITY:
            modFilter = "  " + this.valuefield + " = ?";
            break;
        case TAFilter.BEGINS_WITH:
            modFilter = " " + this.valuefield + " LIKE ?";
            filter.setValue( filter.getValue() + "%" );
            break;
        case TAFilter.ENDS_WITH:
            modFilter = " " + this.valuefield + " LIKE ?";
            filter.setValue( "%" + filter.getValue() );
            break;
        case TAFilter.CONTAINS:
            modFilter = " " + this.valuefield + " LIKE ?";
            filter.setValue( "%" + filter.getValue() + "%" );
            break;
        }

        String query2Check = this.query;

        query2Check = query2Check.toUpperCase();

        if (modFilter.length() > 0) {
            if (query2Check.indexOf(" WHERE ") > 0) {
                this.query = this.query + " AND " + modFilter;

            } else {
                this.query = this.query + " WHERE " + modFilter;
            }
            hasParam = true;
        }//end if (modFilter.length() > 0
    }//end if (null != filter
    final Connection connection = getConnection();
    try {

        pStmt = connection.prepareStatement( this.query );

        if ( LOGGER.isTraceEnabled() ) {
            LOGGER.trace( "query:(" + this.query + "):modfilter:" +
                                                        modFilter );
        }
```

```
            if ( hasParam ) {
                pStmt.setString( 1, filter.getValue() );
            }

            rs = pStmt.executeQuery();
            ResultSetMetaData rsMetaData = rs.getMetaData();
            int nameColumn = 0;
            //Select the first String field whose column name does not
                                // match the value field
            for ( int i = 1; i <= rsMetaData.getColumnCount(); i++ ) {
                if (((rsMetaData.getColumnType(i) == Types.VARCHAR) ||


               (rsMetaData.getColumnType(i) == Types.CHAR))
                       && (!rsMetaData.getColumnName(i).

                equalsIgnoreCase(this.valuefield))) {
                    nameColumn = i;
                    break;
                }//end if
            }//end for

            while ( rs.next() ) {
                SearchResult sr = new SearchResult();

                sr.setValue(rs.getString(this.valuefield));

                if (nameColumn > 0) {
                    sr.setName(rs.getString(nameColumn));
                } else {
                    sr.setName((String) sr.getValue());
                }

                ar.add(sr);
            }
            return ar;
        } catch (SQLException e) {
            LOGGER.fatal("Unable to obtain the values", e);
            TAAttributeDefinitionException exp =
                                            new
              TAAttributeDefinitionException();
               exp.initCause(e);

              throw exp;
        } finally {
            DBTool.close(rs);
            DBTool.close(pStmt);
            DBTool.close(connection);
        }//end finally
        //end getValueConstraint
    }

    /**
     * Gets the Connection to the Database.
     * @return Connection to the database.
     * @throws TAAttributeDefinitionException Thrown if an Exception
           * occurs looking up DataSource, or making Connection
     * to the Database.
     */
    private Connection getConnection() throws
                TAAttributeDefinitionException {
        InitialContext ctx = null;

        try {
            ctx = new InitialContext();
            DataSource ds = (DataSource) ctx.lookup(this.poolname);
            return ds.getConnection();
        } catch (NamingException e) {
            final String msg = "Unable to retrieve Data Connection: " +
                                            e.getMessage();
            LOGGER.fatal(msg, e);
            throw new TAAttributeDefinitionException(msg);
        } catch (SQLException e) {
```

```
            final String msg = "Unable to make Connection to Database: " +
                                                    e.getMessage();
            LOGGER.fatal(msg, e);
            throw new TAAttributeDefinitionException(msg);
        } finally {
            Tools.close(ctx);
        }
        //end getConnection
    }

    /**
     * Convert the Filter (if supplied) into its SQL format, and return
           * the SQL Query.
     * @param query Base SQL Query to apply filter to.
     * @param valuefield
     * @param filter
     * @return
     * @deprecated
     * @see TAFilter#getPreparedStmtStr(String)
     */
    protected static String getFilteredQuery(final String query,
            final String valuefield,
            final TAFilter filter) {
        if ( null != filter ) {
            final StringBuffer modFilter = new StringBuffer(32);
            modFilter.append(' ');
            modFilter.append(valuefield);
            switch ( filter.getOperation() ) {
            case TAFilter.EQUALITY:
                modFilter.append(" = ?");
                break;
            case TAFilter.BEGINS_WITH:
                modFilter.append(" LIKE ?");
                filter.setValue( filter.getValue() + "%" );
                break;
            case TAFilter.ENDS_WITH:
                modFilter.append(" LIKE ?");
                filter.setValue( "%" + filter.getValue() );
                break;
            case TAFilter.CONTAINS:
                modFilter.append(" LIKE ?");
                filter.setValue( "%" + filter.getValue() + "%" );
                break;
            }

            String query2Check = query;

            query2Check = query2Check.toUpperCase();

            if (modFilter.length() > 1) {
                if (query2Check.indexOf(" WHERE ") > 0) {
                     return(query + " AND " + modFilter);
                }// else
                return(query + " WHERE " + modFilter);
            }//end if (modFilter.length() > 0
        }//end if (null != filter
        //else
        return(query);
        //end getFilteredQuery
    }

    private void checkParam( String value, String name ) throws
                TAAttributeDefinitionException {
        if ( ( null == value ) || ( value.trim().length() == 0 ) ) {
            throw new TAAttributeDefinitionException( "The value of " +
                                        name + " is needed" );
        }//end if
        //end checkParam
    }

    /**
     * @see com.hp.ovsi.attribute.SIConstraintIntf#getConstraints(com.hp.
           * ovsi.attribute.api.GetConstraintsCommand)
     */
```

```
public GetConstraintsResponse getConstraints(GetConstraintsCommand
          command) throws TAAttributeDefinitionException{
    //TODO: Normal getConstraintValue
    final TAQuery query = command.getQuery();
    final List filters = query.getTaFilterList();
    //FIXME: This only sends the first filter!
   final TAFilter filter = filters == null || filters.size() < 1 ? null :
   TAFilter)filters.get(0);
    final List constraints = getValueConstraint(command.getAttrName(),
                      filter, command.getParams());
    final GetConstraintsResponse response = new
                      GetConstraintsResponse(constraints);
    return(response);
    //end getConstraints
}

/**
 * @see com.hp.ovsi.attribute.SIConstraintIntf#getDisplayName(com.hp.
      * ovsi.base.Command)
 * INPUT = values
 * OUTPUT = DiaplayValues(Name)/Values pairs
 */
public Response getDisplayName(Command command) throws
      TAAttributeDefinitionException{
    final TAQuery taQuery = command.getQuery();
    final Collection filters = taQuery.getTaFilterList();
    String sqlQuery;
    if (filters != null && filters.size() > 0 ) {
        sqlQuery = this.query + makeWhereClause(filters,
                                    taQuery.isFilterListAnded());
    } else {
        sqlQuery = this.query;
    }
    LOGGER.debug(sqlQuery);
    final Connection connection = getConnection();
    PreparedStatement ps = null;
    ResultSet rs = null;
    try {
        ps = connection.prepareStatement(sqlQuery);
        if (filters != null && filters.size() > 0 ) {
            int idx = 1;
            for (Iterator it = filters.iterator(); it.hasNext();idx++) {
                TAFilter filter = (TAFilter) it.next();
                ps.setObject(idx, filter.getValueAsObject());
            }//end for
        }//end if
        rs = ps.executeQuery();
        final int valCol = rs.findColumn(this.valuefield);
        // The Name column should be either the first or second column
                          // retrieved.
        final int nameCol = rs.getMetaData().getColumnCount() > 1 ?
            (valCol == 1 ? 2 : 1) : 0;
        final boolean namePresent = nameCol > 0;
        final List resultList = new ArrayList(32);
        while(rs.next()) {
            final SearchResult sr = new SearchResult();
            final Object name = namePresent ? rs.getObject(nameCol) :
                                                              null;
            sr.setName(name == null ? null : name.toString());
            final String value = rs.getString(valCol);
            sr.setValue(value == null ? null : value.toString());
            resultList.add(sr);
        }//end while
        return(new Response(resultList));
    } catch (SQLException e) {
        final String msg = "Exception retrieving SQL: " + e.getMessage();
        LOGGER.fatal(msg, e);
        throw new TAAttributeDefinitionException(msg);
    } finally {
        DBTool.close(rs);
        DBTool.close(ps);
        DBTool.close(connection);
    }
```

```
    //end getDisplayName
}

/**
 * Create the SQL Where clause based on the Filters.
 * @param filters The Filters to create the Where clause on.
 * @param anded Whether the filters should be Anded or OR'd together.
 * @return SQL Where Clause.
 */
protected StringBuffer makeWhereClause(final Collection filters,
           final boolean anded) {
    final StringBuffer whereClause = new StringBuffer(1024);
    final StringBuffer inSegment = new StringBuffer(filters.size() * 3);
    for (Iterator it = filters.iterator(); it.hasNext();it.next()) {
        inSegment.append("?,");
    }//end for
    final int length = inSegment.length();
    if (length > 0){
        inSegment.deleteCharAt(length - 1);
        inSegment.append(')');
        if (this.query.toUpperCase().indexOf("WHERE") < 0) {
            whereClause.append(" WHERE ");
        } else {
            whereClause.append(" AND ");
        }
        whereClause.append(this.valuefield);
        whereClause.append(" IN (");
        whereClause.append(inSegment);
    }
    return(whereClause);
    //end makeWhereClause
}

/**
 * @see com.hp.ovsi.attribute.SIConstraintIntf#setParams(java.util.Map)
 */
public void setParams(final Map args) throws
           TAAttributeDefinitionException{
    this.poolname = ( String ) args.get( PARAM_POOLNAME );
    this.query = ( String ) args.get( PARAM_SQLSTRING );
    this.valuefield = ( String ) args.get( PARAM_VALUEFIELD );

    checkParam( this.poolname, PARAM_POOLNAME );
    checkParam( this.query, PARAM_SQLSTRING );
    checkParam( this.valuefield, PARAM_VALUEFIELD );
    //end setParams
}

/**
 * @see com.hp.ovsi.attribute.SIConstraintIntf#size(java.lang.String)
 */
public int size(final String attrName) throws
       TAAttributeDefinitionException {
    return(size(attrName, null));
}

/**
 * @see com.hp.ovsi.attribute.SIConstraintIntf#size(String, TAFilter)
 */
public int size(final String attrName, final TAFilter filter) throws
       TAAttributeDefinitionException {
    if ( LOGGER.isTraceEnabled() ) {
        LOGGER.trace( "Trying to count # of values for attribute:" +
                                        attrName );
    }

    final Connection connection = getConnection();
    PreparedStatement ps = null;
    ResultSet rs = null;
    try {
        ps = connection.prepareStatement(getSizeQuery(filter));
        if (filter != null) {
            ps.setString( 1, filter.getValue() );
        }//end if
```

```
            rs = ps.executeQuery();
            if (rs.next()) {
                return(rs.getInt(1)); //Only one column, the count, returned
            } //else
            throw new TAAttributeDefinitionException("Count not
                                        retrieved.");
        } catch (SQLException e) {
            final String msg = "Exception retrieving Row Count from the
                                        Database: " + e.getMessage();
            LOGGER.fatal(msg, e);
            throw new TAAttributeDefinitionException(msg);
        } finally {
            DBTool.close(rs);
            DBTool.close(ps);
            DBTool.close(connection);
        } //end finally
        //end size
    }

    /**
     * Create the Query to compute the size of the retrieved result set.
     * @param filter Filter to use to filter the results.
     * @return SQL Query based on the supplied filter.
     */
    protected String getSizeQuery(TAFilter filter) {
        if (this.query == null) {
            throw new IllegalStateException("Query not set.");
        }
        String query = this.query.toUpperCase();
        final int fromIdx = query.indexOf("FROM");
        int toIdx = query.indexOf("ORDER");

        if (fromIdx < 0) {
            throw new IllegalStateException("Illegal Query Statement Set:
                                        " + this.query);
        }
        String substr = "";
        if (toIdx > 0){
        substr = this.query.substring(fromIdx, toIdx);
        }
        else {
        substr = this.query.substring(fromIdx);
        }
        query = "SELECT COUNT(*) " + substr ;
        return(getFilteredQuery(query, this.valuefield, filter));
        //end getSizeQuery
    }

    /**
     * @see com.hp.ovsi.Commandable#execute(com.hp.ovsi.base.Command)
     */
    public Response execute(Command command) throws
            UnsupportedOperationException, CommandException {
        final int commandCode = command.getCommand();
        try {
            if (commandCode == GetConstraintsCommand.COMMAND_VALUE) {
            final GetConstraintsCommand gcc =
                                        (GetConstraintsCommand)command;
            setParams(gcc.getParams());
                return(getConstraints((GetConstraintsCommand)command));
            } else if (commandCode == SIConstraintIntf.DISPLAY_COMMAND) {
                try {
                final GetConstraintsCommand gcc =

               (GetConstraintsCommand)command;
                setParams(gcc.getParams());
                    return(getDisplayName(command));
                } catch (TAAttributeDefinitionException e) {
                    throw new CommandException(e.getMessage(), e);
                }
             } else if (commandCode == SIConstraintIntf.SIZE_COMMAND) {
                final GetConstraintsCommand gcc =
```

```
                    (GetConstraintsCommand)command;
                    setParams(gcc.getParams());
                    final TAQuery query = gcc.getQuery();
                    final List filterList = query == null ? null :

                   query.getTaFilterList();
                    final TAFilter filter = filterList == null ? null :
                      filterList.size() > 0 ? (TAFilter)filterList.get(0) : null;
                    final int size = size(gcc.getAttrName(), filter);
                    return(new Response(new Integer(size)));
                }
            } catch (TAAttributeDefinitionException e) {
                throw new CommandException(e.getMessage(), e);
            }
            //else
            throw new UnsupportedOperationException("Command not understood.");
            //end execute
        }

}//end SearchTable
The following is called by the class listed above:
package com.trulogica.truaccess.externalcall.constraintfunction;

import com.trulogica.truaccess.attribute.TAValueConstraintIntf;

public class SearchResult implements
TAValueConstraintIntf.TAValueConstraintBeanIntf {
    private String name;
    private Object value;

    public SearchResult() {
        //Default Constructor }

    public String getName() {
        return this.name; }

    public void setName( String name ) {
        this.name = name; }

    public Object getValue() {
        return this.value; }

    public void setValue( Object value ) {
        this.value = value; }
}
```

Attribute Value Validation External Call

This class determines whether a value is alphanumeric.

```
package com.trulogica.truaccess.externalcall.validation;
import com.trulogica.truaccess.util.logging.misc.Logger;
import com.trulogica.truaccess.util.logging.misc.Level;
import com.trulogica.truaccess.attribute.TAValueValidationIntf;
import com.trulogica.truaccess.attribute.exception.
TAAttributeValueValidationException;
import java.util.*;
public class IsAlphaNumeric implements TAValueValidationIntf
{
     private static final Logger mLogger =
      Logger.getLogger("com.trulogica.truaccess.externalcall.
      validation.IsAlphaNumeric");
     public void validateValue(String attribName,Object value, Map args)
      throws TAAttributeValueValidationException
     {
          String password = (String ) value;
```

```
            if (mLogger.isLoggable(Level.FINE)) mLogger.fine("Entering the
             IsAlphaNumeric method ");
            boolean containsDigit = false;
            boolean containsLetter = false;
            for (int i = 0, n = password.length(); i < n; i++)
            {
                 // checkNum(word.charAt(i));
                 if (Character.isDigit(password.charAt(i)))
                 {
                      containsDigit = true;
                      break;
                 }

            }
            for (int i = 0, n = password.length(); i < n; i++)
            {
                 if (Character.isLetter(password.charAt(i)))
                 {
                      containsLetter = true;
                      break;
                 }
            }
            if (mLogger.isLoggable(Level.FINE)) mLogger.fine("The return of the
             containsDigit is  " + containsDigit);
            if (mLogger.isLoggable(Level.FINE)) mLogger.fine("The return of the
             containsLetter is  " + containsLetter);
            if (containsLetter && containsDigit)
            {
                 if (mLogger.isLoggable(Level.FINE)) mLogger.fine("The external
            call validation is fine. For is digit and alphabets present. ");
            }
            else
            {

                 if (mLogger.isLoggable(Level.FINE))
                 {
                      mLogger.fine("The external call validation is NOT fine.
                 For is digit and alphabets not present. ");
                 }
            throw new TAAttributeValueValidationException(
                 "The field:" + attribName + " is not alpha numeric. It must
                 contain at least one numeric and one non-numeric field",
                 attribName);
            }
       }
}
```

Approver Selection External Call

The following example creates a class that collects information from an existing request and concatenates the information to produce a new value for an attribute. The `getApprover()` function is called by Select Identity to get an approver.

```
package com.trulogica.truaccess.wfenginesvcs.wfexternalcall.support;
import com.trulogica.truaccess.wfengine.wfexternalcall.
 WfSelectApproverIntf;
import com.trulogica.truaccess.wfengine.wfexternalcall.
 WfExternalCallException;
import com.trulogica.truaccess.request.model.RequestTarget;
import com.trulogica.truaccess.util.logging.misc.*;
import java.util.*;
public class WfSelectApproverSample implements WfSelectApproverIntf
{
      /**
       * @param reqtarget is the Request target associated with the workflow
       * @param attrs a list of parameters (standard and custom) for the call
```

```
         * The standard parameters are
         *              serviceid: Internal identifier of the service
         *              servicename: Name of the service
         *              roleid: The role name of the approver
         * The specific parameter is SampleApprovers, which contains comma
         * separated user ids
         *
         * @return The collection of Concero userid of the approver
         * @throws ApprovalStageException
         */
        public Collection getApprover(RequestTarget reqTarget, HashMap attrs)
             throws WfExternalCallException
        {
             ArrayList ret = new ArrayList();
             try
             {
                  String value = (String) attrs.get("SampleApprovers");
                  if (value != null)
                  {
                       StringTokenizer tokens = new StringTokenizer(value, ",");
                       while (tokens.hasMoreTokens())
                       {
                            String user = tokens.nextToken();
                            ret.add(user);
                       }
                  }
             }
             catch (Exception e)
             {
             }
             return ret;
        }
}
```

Workflow Action External Call

Two examples are provided in this section. The first example provides two files. The first file generates an attribute value and adds it to a user, and the second file extends the first by adding an ID. The second example changes a user's attributes and entitlements based on whether the user's department has changed.

Adding a Generated Value to a User

This class generates the new attribute value and adds the value to the user in Select Identity.

```
package com.trulogica.truaccess.wfenginesvcs.wfexternalcall.support;

import com.trulogica.truaccess.wfengine.wfexternalcall.WfExternalCall;
import com.trulogica.truaccess.wfengine.wfexternalcall.AttributeRecord;
import com.trulogica.truaccess.wfengine.wfexternalcall.
     WfExternalCallStatus;
import com.trulogica.truaccess.wfengine.wfexternalcall.
     WfExternalCallException;
import com.trulogica.truaccess.request.model.RequestTarget;
import com.trulogica.truaccess.request.model.RequestTargetParam;
import com.trulogica.truaccess.request.model.RequestTargetParamValue;
import com.trulogica.truaccess.wfengine.wfexternalcall.ChangeRecord;
import com.trulogica.truaccess.base.constants.UserAttributeConstants;
import java.util.*;
//import com.trulogica.truaccess.workflow.external.util.*;

public abstract class WorkflowStepSample implements WfExternalCall
{
```

```
        String attribName = "";

        protected WorkflowStepSample(String _attribName)
        {
             attribName = _attribName;
        }

        public WfExternalCallStatus process(String stageId,
             RequestTarget requestTarget,
             AttributeRecord [] availGrp,
             AttributeRecord [] availRole,
             AttributeRecord [] availEntilements,
             Map map) throws WfExternalCallException
        {
             try{

                  WfExternalCallStatus ecs = new WfExternalCallStatus(stageId);
                  String propPrepName = "", propName = "";

                  propName = attribName + ".attributes";

                  String attrs = System.getProperty(propName);
                  if (null == attrs) {

                  ecs.setStatus(WfExternalCallStatus.STATUS_REJECT_TERMINATE);
                       return ecs;
                  }

                  StringBuffer sb = new StringBuffer();

                  //The default change operation is ADD.

                  ChangeRecord changeRecord = new ChangeRecord();
                  changeRecord.setName(attribName);

                  StringTokenizer st = new StringTokenizer( attrs,"," );
                  while( st.hasMoreTokens() ) {
                       String  attrName = st.nextToken();
                       String value =
                       requestTarget.getParamValueString(attrName);
                       if (null == value) {
                            throw new Exception("Unable to generate the value
                            for attribute:"+attrName);
                       }

                       sb.append(value);
                  }

                  changeRecord.addValue(sb.toString());
                  ecs.addAttributeChange( changeRecord );


                  getSingleRequestParamStr(UserAttributeConstants.FIRST_NAME );

                  ecs.setStatus(WfExternalCallStatus.STATUS_APPROVED);
                  return ecs;
             }

             catch( Exception e ) {
                  throw new WfExternalCallException( e );
             }
        }
}
```

The following class extends the WorkflowStepSample class created above and defines a class that obtains the name of the user.

```
package com.trulogica.truaccess.wfenginesvcs.wfexternalcall.support;
import com.trulogica.truaccess.wfenginesvcs.wfexternalcall.support.
     WorkflowStepSample;

public class PersonNumberCallout extends WorkflowStepSample
{
     public PersonNumberCallout() {
          super("personId");
     }
}
```

Retrieving and Changing Attributes and Entitlements

The following code creates a class the determines whether a user's department (attribute) changed. If the user's department has changed, the class removes entitlements associated with the old department, changes the cost center attribute, and adds new entitlements based on the new department.

```
package com.trulogica.truaccess.wfenginesvcs.wfexternalcall.support;

import java.sql.*;
import java.util.*;
import javax.naming.*;
import javax.sql.*;

import com.trulogica.truaccess.request.model.*;
import com.trulogica.truaccess.util.logging.misc.*;
import com.trulogica.truaccess.wfengine.wfexternalcall.*;

public class WfEntitlementChange
    implements WfExternalCall
{
    private static final Logger mLogger =
          Logger.getLogger(WfEntitlementChange.class.getName());

    public static final String PARAM_POOLNAME = "jdbc/TruAccess";

    //set the resource name here
    private static final String resourceName = "LDAP_70";

    private static final String USERNAMEATTRNAME = "UserName";
    private static final String DEPARTMENTATTRNAME = "Department";
    private static final String COSTCENTERATTRNAME = "CostCenter";
    private static final String ENTITLEMENTPOSTFIX = "_ENTITLEMENTS";

    private static final String findOldDeptQuery = "select A.stringValue
          from TAAttribute A , "
               + "TAUser B where A.identObjId = B.userId AND A.name = "
               + "?  and B.conceroUserId = ?";

    /*
     * If the department has changed, assign the new cost center and
     * entitlements to the user. Use the following table to assign values.
     * Department Entitlements CostCenter
     * ---------- ------------ -----------
     * Sales      SA-505       101
     * Finance    FIN-505      205
     * HR         HR-101       308
     * Corporate  CORP-3       409
     */
    private static final String[] deptNames =
        {
        "Sales", "Finance", "HR", "Corporate"};

    private static final String[] costCenterNames =
```

```
    {
    "101", "102", "103", "100"};

private static final String[] entNames =
                {
    "SA-103", "FIN-101", "HR-102", "Corp-103"};

public WfExternalCallStatus process(String stageId, RequestTarget
      reqTarget, AttributeRecord[] attrbRec1, AttributeRecord[]
      attrbRec2,AttributeRecord[] attrbRec3, Map map) throws
      WfExternalCallException
{
    Connection connection = null;
    PreparedStatement pStmt = null;
    ResultSet rs = null;

    try
    {
        //get service name from map
        //String serviceName =

       (String)map.get(WfExternalCall.WF_PARAM_SERVICENAME);

        WfExternalCallStatus status = new WfExternalCallStatus(stageId);

        //Get the new department and user name from the Request Target
        String newDept =

        reqTarget.getParamValueString(DEPARTMENTATTRNAME);
        String userName =

       reqTarget.getParamValueString(USERNAMEATTRNAME);

        //Get the old department from SI database
        String oldDept = null;
        connection = this.getConnection();

        pStmt = connection.prepareStatement(findOldDeptQuery);
        pStmt.setString(1, DEPARTMENTATTRNAME);
        pStmt.setString(2, userName);

        rs = pStmt.executeQuery();
        if (rs.next())
        {
            oldDept = rs.getString(1);

            if (null == oldDept)
            {
                oldDept = "";
            }
        }
        else
        {
            log("There is no old department");
            oldDept = "";
        }

        //Find whether the department has been changed by the
                          //reconcilation process
        boolean changed = false;
        if (!oldDept.equalsIgnoreCase(newDept))
        {
            changed = true;
            //Build the map tables
        }
        Map entMap = new HashMap();
        Map costMap = new HashMap();

        for (int i = 0; i < deptNames.length; i++)
        {
            entMap.put(deptNames[i], entNames[i]);
            costMap.put(deptNames[i], costCenterNames[i]);
```

```
        }

        //Form the resource attribute name from the resource name.
        //Entitlement attribute names always ends with _ENTITLEMENTS.
        String resourceAttrib = resourceName + ENTITLEMENTPOSTFIX;

        if (changed)
        {
            //Create a change record to add the costcenter value
            //or replace the value if present.
            //CostCenter attribute is a single value attribute.

            ChangeRecord costCenterCR = new ChangeRecord();
            costCenterCR.setName(COSTCENTERATTRNAME);
            if(costMap.get(newDept) != null )
                costCenterCR.addValue( (String) costMap.get(newDept));

            status.addAttributeChange(costCenterCR);

            //Add entitlements to the user
            ChangeRecord entitlementsCR = new ChangeRecord();
            entitlementsCR.setName(resourceAttrib);
            if(entMap.get(newDept) != null )
                entitlementsCR.addValue( (String) entMap.get(newDept));
            /*
             * If you need to add more entitlements, use:
             * cr.addValue("Some other entitlements");
             * To delete attribute values, use:
             * cr.setOperation(ChangeRecord.DELETE);
             */

            //Entitlements are multi-value attributes. need to set change
            //operation ADD/MODIFY/... Default is ADD
            entitlementsCR.setChangeOperation(ChangeRecord.ADD);
            status.addAttributeChange(entitlementsCR);

            //set a work flow variable
            //workflow variable name starts with __
                        //ChangeRecord _wfVar1 = new ChangeRecord();
                        //_wfVar1.setName("__WFVAR");
                        //_wfVar1.addValue("Var value");
        }

        status.setStatus(WfExternalCallStatus.STATUS_APPROVED);
        return status;
    }
    catch (Throwable e)
    {
        log("Unable to complete the department change", e);
        WfExternalCallException exp = new
                    WfExternalCallException("Unable to complete
                    the department change", e);
        throw exp;
    }

    finally
    {
        //Close all database connections, statements and result sets
        try
        {
            if (null != rs)
            {
                rs.close();
            }
            rs = null;
        }
        catch (Throwable t)
        {
            log("Error in closing resultset", t);
        }

        try
        {
```

```
                    if (null != pStmt)
                    {
                        pStmt.close();
                    }
                    pStmt = null;
                }
                catch (Throwable t)
                {
                    log("Error in closing statement", t);
                }

                try
                {
                    if (null != connection && !connection.isClosed())
                    {
                        connection.close();
                    }
                    connection = null;
                }
                catch (Throwable t)
                {
                    log("Error in closing connection", t);
                }
            }
        }

        private Connection getConnection() throws Exception
        {
            InitialContext ctx = null;
            try
            {
                ctx = new InitialContext();
                DataSource ds = (DataSource) ctx.lookup(PARAM_POOLNAME);
                return ds.getConnection();
            }
            finally
            {
                try
                {
                    if (null != ctx)
                    {
                        ctx.close();
                    }
                    ctx = null;
                }

                catch (Throwable t)
                {
                    log("Error closing context", t);
                }
            }
        }

        private void log(String msg, Throwable t)
        {
            System.out.println(msg + ":Caused By:");
            t.printStackTrace();
        }

        private void log(String msg)
        {
            System.out.println(msg);
        }
    }
```

Retrieving Request Object Data to Set Workflow Variables

The following example provides the `WorkflowRequestCallout` class, which looks for a map called `$AttributeResourceMap` (defined in the `MAP_NAME` variable). The map must have a name-value pair whose name is `FieldName` (defined in the `FIELD_NAME` variable) and whose

value is the field to retrieve from the request target (`requestTarget` class). This class also retrieves the value from the request target and stores it in a workflow variable called `WorkflowRequest` (defined in the `WORKFLOW_REQUEST` variable).

```
package com.trulogica.truaccess.workflow.external;

import java.util.ArrayList;
import java.util.Map;
import java.util.HashMap;
import java.util.Set;
import java.util.Iterator;

import com.trulogica.truaccess.wfengine.wfexternalcall.WfExternalCall;
import com.trulogica.truaccess.wfengine.wfexternalcall.AttributeRecord;
import com.trulogica.truaccess.wfengine.wfexternalcall.ChangeRecord;
import com.trulogica.truaccess.wfengine.wfexternalcall.
  WfExternalCallStatus;
import com.trulogica.truaccess.wfengine.wfexternalcall.
  WfExternalCallException;
import com.trulogica.truaccess.request.model.RequestTarget;
import com.trulogica.truaccess.base.constants.UserAttributeConstants;
import com.trulogica.truaccess.util.logging.misc.Logger;
import com.trulogica.truaccess.util.logging.misc.Level;
import com.trulogica.truaccess.wfengine.client.IWfQuery;
import com.trulogica.truaccess.wfengine.client.IWfDataUpdate;
import com.trulogica.truaccess.transport.protocol.ejb.client.
  StatelessServiceObjectFactory;

public class WorkflowRequestCallout implements WfExternalCall
{

    private static final Logger mLogger =
      Logger.getLogger(WorkflowRequestCallout.class.getName());
    public static final String WORKFLOW_REQUEST = "WorkflowRequest";
    public static final String FIELD_NAME = "FieldName";
    public static final String MAP_NAME = "$AttributeResourceMap";

/*
 * Default public constructor.
 */
public WorkflowRequestCallout()
    {
    }

/*
 * Processes the external call request from Select Identity for the person
number.
 *
 * @param stageId                The stage to process
 * @param requestTarget          Object containing fixed roles, groups, and
 *                                 entitlements
 * @param availGrp               Groups that can be assigned to the user
 * @param availRole              Roles that can be assigned to the user
 * @param availEntitlements      Entitlements to be assigned to the user
 * @param attrs                  Additional attributes needed by the call
 *
 * @return  Object containing the person number
 *
 * @exception Throws WFExternalCallException if an error occurs.
 */
public WfExternalCallStatus process(String stageId,
                                    RequestTarget requestTarget,
                                    AttributeRecord [] availGrp,
                                    AttributeRecord [] availRole,
                                    AttributeRecord [] availEntitlements,
                                    Map attrs) throws WfExternalCallException
    {
        try
        {
            WfExternalCallStatus ecs = new WfExternalCallStatus(stageId);
```

```
                IWfQuery query = (IWfQuery)
                   StatelessServiceObjectFactory.create( IWfQuery.class );
                mLogger.finest("WorkflowRequestCallout:  WF_PARAM_WORKFLOWINSTID
                   is " + attrs.get(WF_PARAM_WORKFLOWINSTID));

                Integer workFlowInstanceID = (Integer)
                   attrs.get(WF_PARAM_WORKFLOWINSTID);
                mLogger.finest("WorkflowRequestCallout: workFlowInstanceID is "
                   + workFlowInstanceID);

                int instanceId = workFlowInstanceID.intValue();
                mLogger.finest("WorkflowRequestCallout: instanceId is " +
                   instanceId);

                Map varMap = query.getCurrentVariableMap(instanceId);
                mLogger.finest("WorkflowRequestCallout: varMap is " + varMap);

                HashMap attributeRequestMap = (HashMap) varMap.get(MAP_NAME);

                if (attributeRequestMap != null)
                {
                    String field = (String) attributeRequestMap.get(FIELD_NAME);

                    if (field != null)
                    {
                        IWfDataUpdate du =
                          (IWfDataUpdate)StatelessServiceObjectFactory.
                            create(IWfDataUpdate.class);
                        String value =
                           (String)requestTarget.getSingleRequestParamStr(field);

                       du.setWorkflowVar(instanceId, WORKFLOW_REQUEST, field);

                        mLogger.finest("WorkflowRequestCallout:  Setting status
                            to approved...");

                        // Set the status to approved
                        ecs.setStatus(WfExternalCallStatus.STATUS_APPROVED);
                    }
                }
                else
                {
                    mLogger.severe("WorkflowRequestCallout:  Setting status to
                        rejected...");
                    // Set the status to rejected
                    ecs.setStatus(WfExternalCallStatus.STATUS_REJECT_TERMINATE);
                }

                return ecs;
            }
            catch(Exception e)
            {
                mLogger.severe("WorkflowRequestCallout:  Exception occurred: " +
                   e.getMessage());
                throw new WfExternalCallException( e );
            }
        }
    }
```

AddSecondaryUser External Call

Select Identity provides the add account capability on the Service Subscription page to add a secondary user account to an existing normal or primary account on the same service.

The AddSecondaryUser external call provides the capability to provision a new secondary account when subscribing an existing user to a new service using delegated or self-service `AddNewUser` events. This external call *does not* add the current user account to the service; it creates a secondary user account and designates the existing account as the primary user account.

You can access the source code for the AddSecondaryUser external call from `srcExamples.jar` located on the Select Identity product CD. To use this external call, you must register the call from within Select Identity. See the *HP Select Identity Administration Online Help* for step-by-step instructions.

Implementation Overview

Use the AddSecondaryUser external call *only* when a new secondary account needs to be created for each new service subscription request. *Do not* use this external call to assign services to a primary user account. This call converts the subscribe-to-service request into a secondary account registration request, and therefore creates a secondary account for the primary user.

To use the AddSecondaryUser external call, perform the following steps:

1. Register the **AddSecondaryUser** external call in the Select Identity browser interface. The external call does not require any parameters.
2. Select a service and define delegated add service and/or self add service events in the service role by mapping these events to the appropriate workflow.

   

   A sample workflow template is included in the `srcExamples.jar` file:

   `Workflow+Template_AddSecUserWorkFlow_20070226233751.xml`

   This sample workflow is associated with the AddSecondaryUser external call. It does not define provisioning or post-provisioning blocks, because it leaves the existing user account unchanged and provisions a new secondary account onto the target service.

   For information about workflows and workflow templates, refer to the *HP Select Identity Workflow Studio Online Help*.
3. Associate the **User Generation** function with the **UserName** attribute.
4. Log in to Select Identity as a delegated administrator or end user.
5. Subscribe users to the service selected in step 2 or subscribe yourself to the service.
   - As a delegated administrator, subscribe any existing user account to the service.
   - As an end user, subscribe to the service using the My Identity self-registration page.

   Select Identity creates a new subscribe-to-service request.
6. Make a note of the request ID.

   The workflow instance invokes the AddSecondaryUser external call to create a secondary account on the service identified in step 2 by spawning a new workflow.
7. Check the status of the request ID.

Limitations

This section briefly describes the functional limitations of using the AddSecondaryUser external call.

Limited Service Selection From the Service Subscription Page

A new secondary account can only be assigned to the services that are displayed on the user's Service Subscription page.

This limitation exists only when adding a secondary user account (an `AddAccount` request) from the browser interface. `AddAccount` requests originated via web services or during bulk add or reconciliation operations, are not affected by this limitation.

Multiple Service Registrations by a Single User

This external call does not prevent a single user from registering for the same service multiple times. In this case, multiple secondary accounts are created.

- The AddSecondaryUser external call does not check or validate how many secondary accounts exist for users on subscribed services. However, you can modify the source code to perform this validation.
- If you attach a service associated with this external call to an **Add-to-Service** block in a reconciliation workflow, an additional secondary account is created.

  To prevent additional secondary account creation in reconciliation workflows, perform the following steps:

  a. Modify the external call code to reuse the `GUID` value from the original `requestTarget`.

  b. Change the request event type of the secondary user creation request to `ADD-SERVICE`.

Index

Symbols

A

C

D

E

F

G

I

J

K

L

M

P

Q

R

S

T

U

V

W