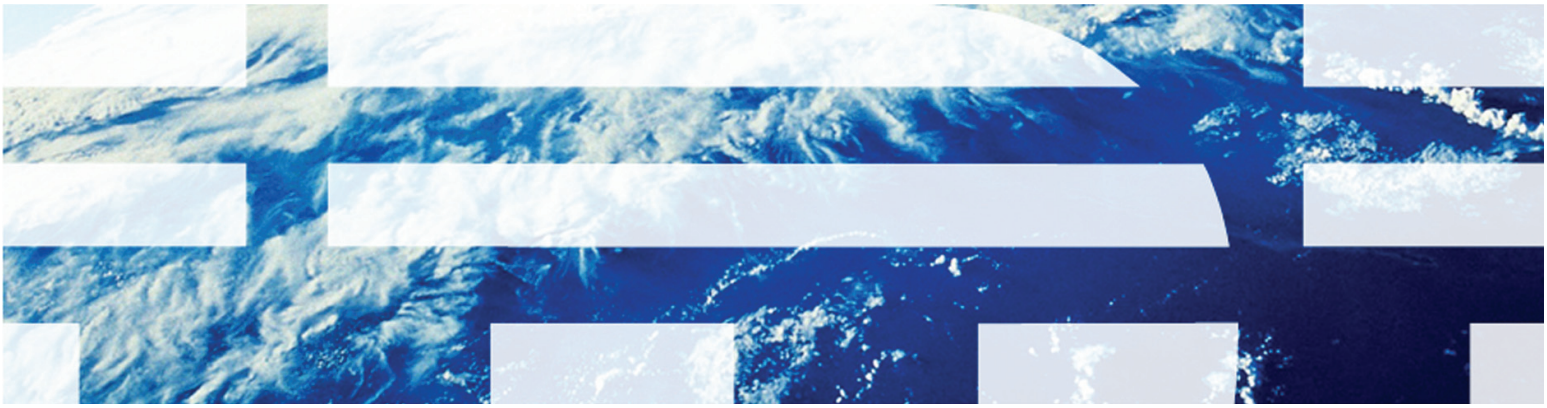

IBM Tivoli Identitiy Manager 5.1: Writing Java Extensions and Application Code

White Paper

April 2012

Ori Pomerantz
orip@us.ibm.com

Table of contents

Introduction

This paper explains how to write Java modules that are called by IBM Tivoli Identity Manager. This way, programmers can extend product functionality and integrate it with other systems.

Audience

This paper is written for Java programmers who need to extend the functionality of IBM Tivoli Identity Manager.

1 Attaching to IBM Tivoli Identity Manager

One method to write an extension in Java that is called from Tivoli Identity Manager (TIM), is to add a new class into the application and then call it from Javascript.

Note: There is an alternative method, which uses classes that implement the **com.ibm.itim.script.ScriptExtension** interface. That method allows extensions to be limited to specific TIM components and access context information such as variables.

That method is also more complicated than the one shown in this white paper. To learn how to use it, look at the demonstration code at **$ITIM_HOME/extensions/5.1/examples/javascript/src/examples/itim50/javascript**

1.1 Adding a new class

TIM is a Java Enterprise Edition (JEE) application, running on top of IBM WebSphere Application Server (WAS). To add a Java class to the application, follow these steps:

1. Compile the Java code into a class file. To use the TIM API, put this file in the class path:

   ```
   $ITIM_HOME/lib/itim_server.jar
   ```

2. Put the class file into a Java archive (JAR) file.
3. Log on to the WAS integrated services console.
4. Create a new shared library that includes the JAR file you created.
5. Modify the **ITIM** application: Change the **Shared library references** (for the whole application rather than a specific module) to add the new shared library.
6. Save the modified WAS configuration.

A demonstration of these steps is available on the IBM Education Assistant site, at URL:

```
http://publib.boulder.ibm.com/infocenter/ieduasst/tivv1r0/
index.jsp?topic=/com.ibm.iea.tim/tim/5.1/
prog_extensions.html
```

1.2 Calling the new class

There are the steps to call the new class from JavaScript:

1. Edit this file:

   ```
   $ITIM_HOME/data/scriptframework.properties
   ```

   In the file, add a property that starts with **ITIM.java.access** whose value is equal to the name of the new class. For example:

   ```
   ITIM.java.access.test=com.ibm.tivoli.orip.TIMAttach
   ```

2. Restart WebSphere Application Server

3. Use the class in JavaScript within TIM to test it. In most cases, the easiest method to do this is to use the class in an identity policy, and request an account.

1.2.1 Simple example

Use these steps to try it out:

1. Compile the **TIMAttach** class in Appendix A and attach it to a testing TIM instance.

2. Create this identity policy:

   ```
   var cls = new com.ibm.tivoli.orip.TIMAttach();
   return subject.getProperty("uid")[0] + cls + cls + cls;
   ```

3. Create a dummy service that uses the identity policy.

4. Request an account for the dummy service. The account form should have the normal user ID for the user, followed by 123.

1.3 Log messages

It is easier to program when you can generate debug messages to see what is happening. To write log messages, use the object **com.ibm.itim.logging.SystemLog**.

This is the syntax to obtain the log writer:

```
com.ibm.itim.logging.SystemLog log =
      com.ibm.itim.logging.SystemLog.getInstance();
```

To actually produce the messages, use the appropriate method to the severity level: **logError**(), **logWarning()**, or **logInformation**():

```
log.logInformation("Java Extension", "Something happened");
```

2 Reading Tivoli Identity Manager entities

This section of the white paper teaches how to read TIM entities from your Java extensions.

Important: The TIM entities are stored in LDAP. You could also read them directly there. However, there is no guarantee that the LDAP data schema would not change in the future. For future portability, you should always use the TIM objects.

2.1 The JavaDocs

The Java API is documented in JavaDocs. To access them, open **$ITIM_HOME/extensions/5.1/api/index.html** with a web browser. The classes that correspond to TIM objects are in the **com.ibm.itim.dataservices.data.model.domain** package.

Figure 1: JavaDocs for the Person class

2.2 Searching for entities

The mechanism to read TIM entities in Java is similar to the one used in JavaScript. First, you create a search object. Then you use a that search object to find a list of data objects which includes the one you need.

For example, to find a person with a specific name, use these steps:

1. Get the search context, the distinguished name at the top of the TIM data structure. If you have multiple searches, you can reuse the same context for all of them.

```
DirectorySystemEntity context =
   new DirectorySystemSearch().lookupDefault();
CompoundDN logicalContext =
   new CompoundDN(context.getDistinguishedName());
```

2. Create a new **PersonSearch** object and use it to search using an LDAP filter and the context you created previously. If necessary, you can specify search parameters, such as the maximum number of entries to return and the maximum query length in time, in the third parameter.

```
SearchResults users =
   new PersonSearch().searchByFilter(
        logicalContext,
        "(cn=" + name + ")",
        new SearchParameters()
   );
```

3. In most cases, it is necessary to check if the search returned any results. Use the **isEmpty()** method.

```
if (users.isEmpty()) {
   log.logWarning("FindMgr", "result is empty");
}
```

4. This line creates a **SearchResultsIterator** out of the **SearchResults**.

```
SearchResultsIterator iter = users.iterator();
```

5. Read a **PersonEntity** from the iterator. Entity objects include methods to modify and delete entities.

```
PersonEntity ent = (PersonEntity) iter.next();
```

6. Get the actual **Person** object. This object contains the actual user information.

```
Person person = (Person) ent.getDirectoryObject();
```

2.3 Reading entity attributes

The entityattributes are accessible using the **getAttribute()** method. These are the steps to access an attribute:

1. Get the **AttributeValue** out of the **DirectoryObject** (such as **Person**).

```
AttributeValue mgrVal = person.getAttribute("manager");
```

2. If the attribute does not exist in the object, the **AttributeValue** is null.

```
if (mgrVal != null)
```

3. If the attribute is single value, use the **getString()** method to get the value.

```
managerDN = mgrVal.getString();
```

4. For multivalued attributes, use the **getAttribute()** method.

```
AttributeValue oc = person.getAttribute("objectClass");
if (oc != null) {
   Object arr[] = oc.getValues().toArray();
}
```

2.4 FindMgr: A full example

The **FindMgr** class in Appendix A takes the name of a user and a number of levels of management, and returns the distinguished name (DN) of the manager at that level. For example, if Eliza reports to David and David reports to Caroline, then the expression:

```
(new com.ibm.tivoli.orip.FindMgr("Eliza", 2)).toString();
```

Evaluates to the DN for Caroline.

A few points in the program require additional explanation.

2.4.1 The try/catch block

```
try {
      ...
} catch (Exception e) {
      log.logError("exception", e.toString());
}
```

Many of the Tivoli Identity Manager functions raise exceptions. The easiest way to handle these exceptions is to generate an error in the log file. A more sophisticated extension would also attempt to perform error recovery.

2.4.2 The default suffix

```
// Get the default suffix to search
DirectorySystemEntity context =
      new DirectorySystemSearch().lookupDefault();
CompoundDN logicalContext =
      new CompoundDN(context.getDistinguishedName());
```

In contrast to JavaScript, in Java the ITIM objects require the ITIM suffix in LDAP to be provided as a parameter. This code snippet shows how to get this suffix.

2.4.3 Following the management chain

```
SearchResults mgrRes =
      new PersonSearch().
      searchByFilter(logicalContext,
           "(erGlobalID=" +
           erGlobalID + ")",
           new SearchParameters());
PersonEntity mgrEnt =
      (PersonEntity) mgrRes.iterator().next();
Person mgr =
      (Person) mgrEnt.getDirectoryObject();
      temp = mgr.getAttribute("manager");
if (temp != null) {
      managerDN = temp.getString();
      erGlobalID = managerDN.substring(11,30);
}
```

The manager's DN is stored in an attribute called **manager**. However, to read a **Person** object from the directory it is necessary to search for it. Distinguished names of users in TIM have this form:

```
erglobalid=3026648537936938403,ou=0,ou=people,erglobalid=000
00000000000000000,...
```

The easiest way to get a **Person** object from a DN is to search for the **erGlobalID**, as this code does. It first searches for the manager, using the previous **erGlobalID** attribute. Then, to get the manager's manager, it reads the **manager** attribute and isolates the **erGlobalID** out of the DN.

3 Modifying Tivoli Identity Manager entities

The API to modify TIM entities uses object classes called **<*name of entity*>MO**. For example, to modify accounts, instantiate the **AccountMO** class.

3.1 Connecting to Tivoli Identity Manager

There is no specific API for modifying TIM entities internally from within WebSphere Application Server. To modify them, it is necessary to use an external API, which is designed for modules that run separately. Therefore, it is necessary to connect to that API and authenticate the connection.

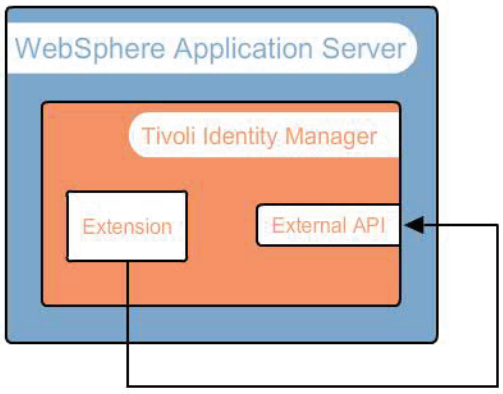

Figure 2: An extension connecting through the external API

Important: The TIM entities are stored in LDAP. You could also read them directly there. However, there is no guarantee that the LDAP data schema would not change in the future. For future portability, you should always use the TIM objects.

3.1.1 Creating the InitialPlatformContext

The first step is to specify the communication parameters, which are stored in an **InitialPlatformContext** object. Those parameters are available in the properties file **$ITIM_HOME/data/enRole.properties** as **enrole.appServer.contextFactory** and **enrole.appServer.url**.

To get TIM properties, use a **com.ibm.itim.common.properties.PropertiesManager** object. The **getProperties()** method takes two arguments, both property names. TIM looks for the first property name in **Properties.properties** to locate the property file. For example, the property **enrole** has the value **enRole.properties**. In that file, TIM looks for the second property name, and returns the corresponding value.

```
// The platform context environment variables
// come from, $ITIM_HOME/data/enRole.properties.
Hashtable<String, String> env =
          new Hashtable<String, String>();
PropertiesManager pm =
          PropertiesManager.gInstance();
env.put(InitialPlatformContext.CONTEXT_FACTORY,
          pm.getProperties("enrole",
               "enrole.platform.contextFactory"));
env.put(InitialPlatformContext.PLATFORM_URL,
          pm.getProperties("enrole",
               "enrole.appServer.url"));
InitialPlatformContext context =
          new InitialPlatformContext(env);
```

3.1.2 Logging on

Because this API is exposed over TCP/IP, connections have to be authenticated with the user name and password of the user whose permissions are used for the requests. This code logs on as the administrator.

```
String itimUser = "ITIM Manager";
String itimPswd = "object00";

// Create the ITIM JAAS CallbackHandler
PlatformCallbackHandler handler =
      new PlatformCallbackHandler(itimUser,
                                  itimPswd);
handler.setPlatformContext(context);

// Associate the CallbackHandler with a LoginContext,
// then try to authenticate the user with the platform
LoginContext lc = new LoginContext("ITIM", handler);
lc.login();
```

3.2 Modifying the entity

To actually modify data requires four steps:

1. Create an **<*name of entity*>MO** object using the context, the connection you created earlier, and the DN for the entity to be modified.
2. Use the **getData** method to retrieve the data object for the entity. This object always belongs to a subclass of **DirectoryObject**.
3. Modify the entity. You can use the object attribute manipulation methods of **DirectoryObject** (**addAttribute**, **removeAttribute**, and **setAttribute**). Alternatively, you can use the class specific methods, such as **Account.setPassword** or **Person.removeRole**.
4. Use the **update** method of the **<*name of entity*>MO** object to update the data. The second parameter contains the time for the change if it is not immediate.

This code fragment shows those four steps. It removes the **manager** attribute from a person. The full program is **RemoveMgr.java** in Appendix A.

```
// Remove the manager attribute
PersonMO personMO = new PersonMO(context,
                        lc.getSubject(),
                         new DistinguishedName(dn));
Person data = personMO.getData();
data.removeAttribute("manager");
personMO.update(data, null);
```


4 Creating new Tivoli Identity Manager entities

To create new TIM entities, connect to the external TIM API as you do for modifications. Then follow these steps to actually create the entity:

1. Create an **<*name of entity*>Manager** object. This object is used to create new entities.

```
// Create the PersonManager object
PersonManager mgr = new PersonManager(context,
     lc.getSubject());
```

2. TIM entities exist in a location in the organization tree. The next step is to create an **OrganizationalContainerMO** object for that location.

```
// Creating new persons requires an
 // OrganizationalContainerMO
OrganizationalContainerMO containerMO = new
 OrganizationalContainerMO(context,
                lc.getSubject(),
                new DistinguishedName(container));
```

3. The next step is to create an object to hold the new entity and add the attributes. Note that some constructors require the entity type as a parameter.

```
// Create a Person object
Person added = new Person("Person");
added.addAttribute(new AttributeValue("cn",
     "Test User"));
added.addAttribute(new AttributeValue("sn",
     "User"));
added.addAttribute(new AttributeValue("uid",
     "test"));
```

4. Finally, use the **<*name of entity*>Manager** to create the new entity.

```
mgr.createPerson(containerMO, added, null);
```

4.1 CreatePerson

The **CreatePerson** class in Appendix A shows a complete program that creates a new Person object called **Test User**.

4.1.1 Calling CreatePerson from JavaScript

The constructor expects to get the distinguished name of the location in the organization tree for the new person (also called the container). This distinguished name is stored in the **erparent** attribute of the person after creation.

For example, to create a new person in the same container as an existing Person for debugging purposes use this JavaScript:

```
new com.ibm.tivoli.orip.CreatePerson(
      subject.getProperty("erparent")[0]);
```


Appendix A Source code

The source code in this appendix is available on the IBM Education Assistant site:

http://publib.boulder.ibm.com/infocenter/ieduasst/tivv1r0/index.jsp?topic=/com.ibm.iea.tim/tim/5.1/prog_extensions.html

A.1 TIMAttach.java

```
package com.ibm.tivoli.orip;

public class TIMAttach
{
      Integer num;

      public TIMAttach()
      {
           num = 0;
      }

      public String toString()
      {
           num++;
           return num.toString();
      }
}
```

A.2 FindMgr.java

```
package com.ibm.tivoli.orip;

import com.ibm.itim.dataservices.model.domain.*;
import com.ibm.itim.dataservices.model.*;
import com.ibm.itim.common.*;

public class FindMgr
{
    String managerDN;
    String erGlobalID;

    public FindMgr(String name, int level)
```

```
{
    com.ibm.itim.logging.SystemLog log =
        com.ibm.itim.logging.SystemLog.getInstance();

    try {
        AttributeValue temp;
        // Get the default suffix to search
        DirectorySystemEntity context =
            new DirectorySystemSearch().lookupDefault();
        CompoundDN logicalContext =
            new
           CompoundDN(context.getDistinguishedName());

        // Search for the user
        SearchResults users =
            new PersonSearch().
                searchByFilter(logicalContext,
                "(cn=" + name + ")",
                new SearchParameters());

        if (users.isEmpty()) {
           log.logWarning("FindMgr", "result is empty");
        }

        SearchResultsIterator iter = users.iterator();
        PersonEntity ent = (PersonEntity) iter.next();
        Person person = (Person)
                ent.getDirectoryObject();
        temp = person.getAttribute("manager");
        if (temp != null) {
            managerDN = temp.getString();
            erGlobalID = managerDN.substring(11,30);
        }

        if (level > 1) {
            for (int i=1; i<level; i++) {
                SearchResults mgrRes =
                    new PersonSearch().
                    searchByFilter(logicalContext,
                                   "(erGlobalID=" +
                                      erGlobalID + ")",
                                new SearchParameters());
                PersonEntity mgrEnt =
                    (PersonEntity)
                        mgrRes.iterator().next();
                Person mgr =
                    (Person)
                        mgrEnt.getDirectoryObject();
                temp = mgr.getAttribute("manager");
                if (temp != null) {
                    managerDN = temp.getString();
                    erGlobalID =
                        managerDN.substring(11,30);
                }
```



```
                }
            }
        } catch (Exception e) {
            log.logError("exception", e.toString());
        }
    }

    public String toString()
    {
        return managerDN;
    }
}
```

A.3 RemoveMgr.java

```
package com.ibm.tivoli.orip;


import java.util.Hashtable;
import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;


import com.ibm.itim.dataservices.model.domain.*;
import com.ibm.itim.dataservices.model.*;
import com.ibm.itim.common.*;
import com.ibm.itim.apps.InitialPlatformContext;
import com.ibm.itim.apps.identity.PersonMO;
import
  com.ibm.itim.apps.jaas.callback.PlatformCallbackHandler;
import com.ibm.itim.common.properties.PropertiesManager;



public class RemoveMgr
{
    public RemoveMgr(String dn)
    {
      com.ibm.itim.logging.SystemLog log =
          com.ibm.itim.logging.SystemLog.getInstance();

      try {

          // The platform context environment variables
          // come from, $ITIM_HOME/data/enRole.properties.
          Hashtable<String, String> env =
               new Hashtable<String, String>();
          PropertiesManager pm =
               PropertiesManager.gInstance();
          env.put(InitialPlatformContext.CONTEXT_FACTORY,
```

```
                    pm.getProperties("enrole",
                         "enrole.platform.contextFactory"));
               env.put(InitialPlatformContext.PLATFORM_URL,
                    pm.getProperties("enrole",
                         "enrole.appServer.url"));
               InitialPlatformContext context =
                    new InitialPlatformContext(env);

               // On a production system, these would be
               // properties
               String itimUser = "ITIM Manager";
               String itimPswd = "object00";

               // Create the ITIM JAAS CallbackHandler
               PlatformCallbackHandler handler =
                new PlatformCallbackHandler(itimUser,
                                  itimPswd);
               handler.setPlatformContext(context);

               // Associate the CallbackHandler with a
               // LoginContext, then try to
               // authenticate the user with the platform
               LoginContext lc = new LoginContext("ITIM",
                    handler);
               lc.login();

               // Find and modify the user
                 PersonMO personMO = new PersonMO(context,
                                  lc.getSubject(),
                                   new DistinguishedName(dn));
               Person data = personMO.getData();
               data.removeAttribute("manager");
               personMO.update(data, null);
          } catch (Exception e) {
               log.logError("exception in RemoveMgr",
                    e.toString());
          }

     }
}
```

A.4 CreatePerson.java

```
package com.ibm.tivoli.orip;


import java.util.Hashtable;
import javax.security.auth.Subject;
```



```
import javax.security.auth.login.LoginContext;



import com.ibm.itim.dataservices.model.domain.*;
import com.ibm.itim.dataservices.model.*;
import com.ibm.itim.common.*;
import com.ibm.itim.apps.InitialPlatformContext;
import com.ibm.itim.apps.identity.PersonManager;
import
com.ibm.itim.apps.jaas.callback.PlatformCallbackHandler;
import com.ibm.itim.apps.identity.OrganizationalContainerMO;
import com.ibm.itim.common.properties.PropertiesManager;



public class CreatePerson
{
    String retval;

    // Create a person called Test User in the org tree
    // location provided as a parameter
    public CreatePerson(String container)
    {
      com.ibm.itim.logging.SystemLog log =
          com.ibm.itim.logging.SystemLog.getInstance();

      try {
          // The platform context environment variables
          // come from, $ITIM_HOME/data/enRole.properties.
          Hashtable<String, String> env =
               new Hashtable<String, String>();
          PropertiesManager pm =
               PropertiesManager.gInstance();
          env.put(InitialPlatformContext.CONTEXT_FACTORY,
              pm.getProperties("enrole",
                   "enrole.platform.contextFactory"));
          env.put(InitialPlatformContext.PLATFORM_URL,
              pm.getProperties("enrole",
                   "enrole.appServer.url"));
          InitialPlatformContext context =
               new InitialPlatformContext(env);

           // On a production system, these would be
           // properties
          String itimUser = "ITIM Manager";
          String itimPswd = "object00";

          // Create the ITIM JAAS CallbackHandler
          PlatformCallbackHandler handler =
           new PlatformCallbackHandler(itimUser,
                            itimPswd);
          handler.setPlatformContext(context);
```

```
                // Associate the CallbackHandler with a
                // LoginContext, then try to authenticate
                // the user with the platform
                LoginContext lc = new LoginContext("ITIM",
                     handler);
                lc.login();

                // Create the PersonManager object
                PersonManager mgr = new PersonManager(context,
                     lc.getSubject());

                // Creating new persons requires an
                // OrganizationalContainerMO
                OrganizationalContainerMO containerMO = new
                 OrganizationalContainerMO(context,
                                lc.getSubject(),
                                new DistinguishedName(container));

                // Create a Person object
                Person added = new Person("Person");
                added.addAttribute(new AttributeValue("cn",
                     "Test User"));
                added.addAttribute(new AttributeValue("sn",
                     "User"));
                added.addAttribute(new AttributeValue("uid",
                     "test"));

                mgr.createPerson(containerMO, added, null);


                retval = container;
              } catch (Exception e) {
                log.logError("exception in CreatePerson",
                     e.toString());
              }
            }

            public String toString() {
              return retval;
            }
        }
```

