SIEMENS

Teamcenter 11.2 Systems Engineering and Requirements Management

Systems Architect/ Requirements Management API Reference

REQ00006 • T

Manual History

Manual Revision	Teamcenter Requirements Version	Publication Date
A	4.0	December 2003
B	4.1	February 2004
C	5.0	July 2004
D	6.0	March 2005

Manual Revision	Teamcenter Systems Engineering and Requirements Management Version	Publication Date
E	2005	September 2005
F	2005 SR1	June 2006
G	2007	December 2006
H	2007.1	April 2007
I	2007.2	September 2007
J	2007.3	January 2008
K	8	January 2009
L	8.0.1	June 2009
M	8.1	October 2009
N	8.2	October 2010
O	9	July 2011
P	9.1	May 2012
P1	9.1.5	January 2014
Q	10.0	January 2015
R	10.1	September 2016

Manual Revision	Teamcenter Systems Engineering and Requirements Management Version	Publication Date
S	11.1	March 2018
T	11.2	September 2019

This edition obsoletes all previous editions.

API Reference Contents

Preface

This manual is an API reference for Teamcenter Systems Architect/Requirements Management . Systems Architect/Requirements Management belongs to the Siemens PLM Software portfolio of digital product lifecycle management software and services.

Audience

This manual contains information for Systems Architect/Requirements Management developers. It assumes that you understand the structure of the Systems Architect/Requirements Management product and are familiar with the installation and maintenance of Systems Architect/Requirements Management.

Conventions

This manual uses the conventions described in the following sections:

Names and Values

This manual represents system names, file names, and values in fonts that help you interpret the name or value. For example:

Change or add the parameter to the **init***sid***.ora** file.

The conventions are:

Bold	Bold font represents unvarying text or numbers within a name or value. Capitalization is as it appears.
Italic	Italic font represents text or numbers that vary within a name or value. The characters in italic text describe the entry. Letters are shown in lowercase, but the varying text may include uppercase letters. In **init***sid***.ora**, *sid* identifies a varying portion of the name (a unique system ID). For example, the name of the file might be: **initBlue5.ora**
text-text	A hyphen separates two words that describe a single entry.

Command Line Entries, File Contents, and Code

This manual represents command line input and output, the contents of system files, and computer code in fonts that help you understand how to enter text or to interpret displayed text. For example, the following line represents a command entry:

```
msqlora -u system/system-password
```

The conventions are:

`Monospace`	Monospace font represents text or numbers you enter on a command line, the computer's response, the contents of system files, and computer code. Capitalization and spacing are shown exactly as you must enter the characters or as the computer displays the characters.
`Italic`	Italic font represents text or numbers that vary. The words in italic text describe the entry. The words are shown in lowercase letters, but the varying text may include uppercase letters. When entering text, use the case required by the system. For the preceding example, you might substitute the following for *system-password*: `KLH3b`
`text-text`	A hyphen separates two words that describe a single entry.

Submitting Comments

Portions of Teamcenter software are provided by third-party vendors. Special agreements with these vendors require Siemens PLM Software to handle all problem reports concerning the software they provide. Please submit all comments directly to Siemens PLM Software.

Please feel free to share with us your opinion on the usability of this manual, to suggest specific improvements, and to report errors. Mail your comments to:

Siemens PLM Software Technical Communications
5939 Rice Creek Parkway
Shoreview, MN 55126
U.S.A.

To submit an incident report, you can use the Siemens PLM Software GTAC online support tools at the following URL:

http://www.plm.automation.siemens.com/en_us/support/gtac/

Proprietary and Restricted Rights Notice

Chapter 1: Introduction

This chapter contains introductory information about the Systems Architect/Requirements Management application programming interface (API).

The Systems Architect/Requirements Management API provides developers with a consistent and stable interface allowing programmatic access to the information within Systems Architect/Requirements Management. The developer has the option of implementing the API at a high level using Java™ or by using Tcl scripting.

- **Java Access:** The functions in the **RequirementService** class are available for use when writing programs in Java or JavaScript. The database contents can be read, modified, or updated programmatically without using the client GUI (graphical user interface). All functions that are available from the GUI can be accessed using Java, without the GUI interface.
- **Tcl Scripting Access:** The database can be read, modified or updated programmatically using Tcl (Tool Command Language) scripts while using the GUI. You can add activators if you have project administrators rights. These activators can be triggered on the occurrence of one or more events such as Create, After Delete, After Modify, Before Delete, and Before Modify on the BuildingBlocks, Folders, Notes, Requirements object types and their subtypes.
- **C# Access:** Server calls can be made from a COM client such as VBA (Visual Basic for Applications) macro using the C# API.

While both Java and Tcl provide full access to the Systems Architect/Requirements Management API, there are differences in how database transactions are handled. In Java, each API call runs in a separate transaction. In Tcl, all the API calls made within a Tcl script run in the same transaction. Because Tcl can run within a single transaction, it is the preferred method to use for iterative or large batch applications.

The two APIs can be used together very effectively in certain situations. A Java API caller can (through the runActivator function) run an activator's Tcl code. The Java program can remain in charge of an overall process, while handing off its iterative actions to Tcl running within the scope of a single transaction. This is much, much more efficient than making repetitive calls through the Java API.

The list of Java API functions is provided in the Javadocs. The Javadocs describe each function along with the response expected from the server. To browse a full list of functions, see the Javadoc HTML documentation, installed with Systems Architect/Requirements Management on the Web server. The exact URL may vary, depending on how Systems Architect/Requirements Management was installed in your web server, but the Javadoc URL will be similar to this:

http:// ... /tcr/ugs/tc/req/apidoc/index.html

The C# API provides an interface that allows making server calls from .Net code and an interface that exposes the API methods to COM.

Chapter 2: API Overview

This chapter contains standard input parameters and lists common API functions.

Standard Input Parameters

Each function takes a set of input parameters unique to the function. There are also several input parameters that apply to a wide range function calls. Table 2-1 lists the standard input parameters.

Table 2-1. Standard Input Parameters

Input	Description
User	Specifies the user session ID. This is the first parameter for almost all Java API methods. It uniquely identifies an existing (already logged in) user session. It consists of the user's login name concatenated with the HTTP session ID.
Properties	Specifies the property values to populate in returned data beans. This can consist of actual property names or one of the constants in **com.edsplm.tc.req.databeans.DataBean** (USER_PROPERTY, SYSTEM_PROPERTY, ALL_PROPERTY or FULL_PROPERTY).

Listing of API Methods

The complete list of API methods is shown in the table below. The table describes each method and indicates if the method is available to the Java API or the Tcl Scripting API.

Table 2-2. Available API Methods

Method	Description	Java	C#	Tcl
authenticate SSOUser	Validates a user name against single sign-on server using an SSO session key.	X		
authenticateUser	Validates a user name and password.	X		
calculate Properties	Calculates the numeric properties in objects.	X		X
changeApproval	Updates the status of change approval objects.			X
copyObjects	Copies the list of objects to the given destination.	X		X
createAction	Instruct Systems Architect/Requirements Management client to launch a URL or run a Macro.			X
createAction FileDownload	Downloads a file from the server to the Systems Architect/Requirements Management client workstation, and optionally, open it in an application. The C# method name is **downloadfileCOM**.		X*	X
createBaseline	Creates a baseline containing the specified objects.	X		X
createExternalLink	Creates a Teamcenter Interface (WOLF) link from Systems Architect/Requirements Management to an object in an external application such as Teamcenter Engineering or Teamcenter Enterprise.	X		X
createLinks	Creates Trace Links between fromID object and to each object in toIDs.	X	X	X
createObject	Creates a new object in the database as a LAST_MEMBER or FIRST_MEMBER or LAST_SIBLING or NEXT_SIBLING.	X	X	X

Table 2-2. Available API Methods

Method	Description	Java	C#	Tcl
createProject	Creates a new project in the database.	X		X
createShortcuts	Creates a new shortcut to the existing object. The C# method name is **createShortcut**.	X	X*	X
createUser	Creates a user object given user name. Also, sets the password and the userâ€™s maximum privilege for the database.	X		X
createVariant	Creates a variant of a versionable object.	X		X
createVersion	Creates a version of a versionable object.	X	X	X
deleteLinks	Deletes complying or defining traceLinks between fromID object and objects in toIDs.	X	X	X
deleteObjects	Deletes the list of objects and moves it to trash can of the caller.	X	X	X
displayMessage	Displays a message in the Systems Architect/Requirements Management client.			X
emptyTrashcan	Empties this users trash can by actually destroying the objects from the database.	X		X
export2Excel	Export an Microsoft Excel file.			X
exportDocument	Writes objects from the database to a Word, Excel, AP233, or XML file.	X	X	X
exportXML	Exports Systems Architect/Requirements Management schema objects or projects to an XML file.			X
getAdminMap	Gets the specified administration module folder from the database	X		
getEnvironment	Retrieves one of Architect/Requirementsâ€™s configuration parameters.	X		X

Table 2-2. Available API Methods

Method	Description	Java	C#	Tcl
getList	Returns a list of Systems Architect/Requirements Management objects related to the given object.	X	X	X
getObject	Gets the given object (one) from the database with set of properties.	X	X	X
getObjects	Gets the given objects (one or more) from the database with set of properties.	X	X	
getProjects	Returns the list of projects you have access to and returns a specified set of properties for each object on the list.	X		X
getProperties WithFormula	Gives NumericPropertyDefinitions that have a formula attached for the passed objects.	X		X
getProperty Definition	Gets the property definition with the name propName from the project objID.	X	X	X
getProperty Definitions	Gets property definitions from the database.	X	X	X
getRemote ObjectTrace Report	Retrieves the trace report for the remote object.	X		X
getType Definition	Gets the type definition with the name **propName** from the project **objId**.	X		
getValue	Gets the value of a property from the given object.	X		X
importDocument	Imports a document of type XML, AP233, Excel, or Word. The C# method name is **importFile**.	X	X*	X
logoutSSOUser	Logs out the given user and closes the Systems Architect/Requirements Management and SSO sessions.	X		
logoutUser	Logs out the user whose user session ID is passed in.	X		

Table 2-2. Available API Methods

Method	Description	Java	C#	Tcl
moveObjects	Moves the list of objects to a destination object.	X	X	X
releaseObject	Releases reservation on an object.	X	X	
restoreFrom Trashcan	Restores objects from this user's trash can to the old owner.	X	X	X
runActivator	Runs the specified activator specified by Activator ID.	X	X	X
runActivator	Runs the specified activator, activator specified by activator name in the specified project.	X		X
runReport	Searches the database for objects that match the specified criteria and output them in a report file.	X	X	X
runScript	Runs a Tcl script.	X		
search	Return a list of objects matching the search parameters.			X
search	Searches the database for objects that match the specified criteria. The search is not case sensitive.	X	X	X
sendEmail	Tcl command class to send E-Mail to List of Email Ids.	X	X	X
setEnvironment	Sets one of Systems Architect/Requirements Management â€™s configuration parameters.			X
setObject	Sets the given object properties and the passed in value in the database the setObject requires NO response.	X		
setObject	Sets the given object properties in the database. If setObject requires a response, set a response.	X		X
setObjects	Sets the given objects' properties in the database.	X	X	
	Sets password for a user.	X		X

Table 2-2. Available API Methods

Method	Description	Java	C#	Tcl
setPassword				
setUser Preferences	Allows a user specify their location so information can be formatted appropriately.	X		X
setValue	Sets the value of a property for a given object. This function can be used to set one value of the object (unlike setObject that sets multiple values).			X
undo	Undoes the last database modification.	X	X	
uncoupleShortcuts	Uncouples a shortcut from the master object, creating a copy of the master object.	X		X
writeLog	Writes a message to the server log file.			X

* Indicates the C# method name is different. See description for the method.

Schema Object Display Name and their Actual Name

In some instances, the name displayed in the Systems Architect/Requirements Management client for out of the box schema objects, such as property and type names, does not match the name stored in the database. This is because the Systems Architect/Requirements Management client can map the actual name to something else. This mechanism is primarily intended for mapping the names to different languages. But some names are also mapped in English.

For example, the **Data Definition** and **Data Dictionary** type names are stored in the database with no space, **DataDefinition** and **DataDictionary**. However, for API calls, you must use the actual schema object name as the display name does not work. This can cause confusion when schema object names are used in API methods such as **createObject** and **setValue**. You can determine the actual name of a schema object by creating a macro with the content:

```
displayMessage [getValue $selected Name]
```

This displays the actual name of the selected schema object.

Chapter 3: Using the Java API Client Library to Access the API

This chapter contains information on using the Java API client library to access the API. If you are developing your client application using Tcl (Tool Command Language) scripting, you can use the Tcl client library included in the toolkit.

Using the API

The **RequirementService** class serves as Systems Architect/Requirements Management Enterprise API. **RequirementService** contains methods that provide complete access to the Systems Architect/Requirements Management server and database. All methods in **RequirementService** are static, so no instance of the class is required. The RequirementService class is located in the **com.edsplm.tc.req.enterprise** package. It is recommended that the Javadocs for **RequirementService** as well as for the package **com.edsplm.tc.req.databeans** be reviewed thoroughly.

A typical single transaction involves, obtaining a session ID, making a call using one of the API functions listed in **RequirementService**, obtaining a **ResultBean** from the returning API call, and checking to see if your transaction was successful by inspecting the **ResultBean** objects.

Logging In to the API

To access the Systems Architect/Requirements Management API you must login using the **AuthenticateUser** or **AuthenticateSSOUser** method. A session ID is constructed from the unique ID that is passed to a successful call to **AuthenticateUser** or **AuthenticateSSOUser**. The session ID takes the form of the login user name concatenated with the unique ID that was passed to **AuthenticateUser** or **AuthenticateSSOUser**. The unique ID is usually obtained by asking the J2EE server for the HttpSession ID. Once a successful login occurs, Systems Architect/Requirements Management keeps track of the user session identified by the session ID. From this session ID, all subsequent API calls must include the session ID as the first argument.

User sessions can be closed by calling **logoutUser** or **logoutSSOUser**. If a logout is not expressly called, the user session times out.

```
// Login to Teamcenter Requirements
ResultBean aBean = RequirementService.authenticateUser("username",
                   "password",  HTTPSessionID, null);
// Build session ID to use for each API call
String SessionID = "username" + HTTPSessionID;

// API calls...

RequirementService.logoutUser("username", HTTPSessionID);
```

Describing the API Functions

The API functions range from creating objects to creating users to searching. To browse a full list of functions, see the Javadoc HTML documentation, installed with Systems Architect/Requirements Management on the web server. The exact URL may vary, depending on how Systems Architect/Requirements Management was installed in your web server, but the Javadoc URL will be similar to this:

http:// ... /tcr/ugs/tc/req/apidoc/index.html

Describing ResultBeans

All the **RequirementService** API methods return a **ResultBean**. The result bean contains a flag indicating if the call succeeded. If no error occurred, the **ResultBeans** contain the desired return value for the API call, typically a **DataBean** or vector of **DataBeans** that represent Systems Architect/Requirements Management objects or strings. The **ResultBean** class is located in the **com.edsplm.tc.req.databeans** package.

In addition a result bean contains a message list, a change list, and a schema list.

Message List and Error Handling

The **MessageBean** class is located in the **com.edsplm.tc.req.databeans** package. A message bean contains the information for displaying a single error, warning, or information message to the user. The bean contains a unique tag name for the message which can be retrieved with the **getTag** method. The tag names are brief descriptions of the issue. For example, **PASSWORD_INCORRECT** tag is used when the wrong password is entered during login. The tag is converted to a readable message on the client using a language bundle. The message lookup service is not available to server APIs.

In addition to the tag, a message bean can contain substitutions and appended text. Substitutions are variable information included within a message. They can be retrieved with the **getSubstitutions** method. Appended text is displayed after the message. Substitutions are typically used to show the database objects involved. The **getAppendedText** method is used to get the information on the appended text.

A **ResultBean** contains a list of any messages generated during the API call. If an error occurs, the message list contains an error message to indicate the problem. In addition to the error message, the entire transaction is nullified.

```
// Get a list of all the TCR projects
ResultBean resBean = RequirementService.getProjects(SessionID,
                     null, null);
If (resBean .isSuccess()) {
   // call succeeded, get the result
   Vector result = resBean.getResult();
} else {
   // call failed, get the error messages
   Collection messages = resBean.getMessageList();
   for (Iterator i = messages.iterator(); i.hasNext(); ) {
          MessageBean msg = (MessageBean)i.next();
          System.out.println(msg.getTag());
   }
}
```

Change List

For API calls that modify Systems Architect/Requirements Management objects, a change list is returned to indicate the new state of any modified objects. A data bean for each modified object is contained in the change list. The change list is useful for refreshing objects in a user interface. Because API callers commonly do not need the change list, performance can be improved by turning this feature off during login.

```
Properties properties = new Properties();
properties.setProperty(DataBean.SESSION_PROPS.RETURN_CHANGES, "false");
RequirementService.authenticateUser("username", "password",
                                    HTTPSessionID, properties);
```

Schema List

ResultBeans also may contain schema information describing the returned objects. The schema list is only returned for **getObject** and **getObjects** methods. The schema list contains data beans for property definitions. A property definition for each requested property is included. Property definitions give detailed information about a property such as the complete choice list for a choice property.

Database Transactions

Every requirement service call uses its own database transaction. When an error occurs, the transaction is cancelled and any changes are rolled back. If the transaction succeeds, changes are committed to the database. The undo method can be used to roll back the changes from prior requirement service calls. Since each requirement service call is a database transaction, transaction management can cause performance problems when large numbers of API calls are made.

When calling a large number of API methods, you should consider using the Tcl API. The Tcl API is an internal API so it can run as a single transaction.

Internal vs External Java API

There are two types of Java APIs, external and internal. Both external and internal APIs have almost exactly the same set of methods. However, their arguments differ slightly as every external call has to carry **sessionID** information whereas the internal calls occur inside a transaction where the session is already known and authenticated.

The **RequirementService** class is the external Java API. Every call is like a request coming from a Systems Architect/Requirements Management client. For each **RequirementService** call such as **getValue**, or **deleteObject**, the Systems Architect/Requirements Management server performs the following:

1. Validate the user **sessionID** passed.
2. Find and connect to an available session in the Versant session pool.
3. Run any before activators that are appropriate as required by the incoming call.
4. Perform the action of the **RequirementService** call.
5. Run any after activators, based on the objects that have changed.
6. Assemble a result bean with its change beans.
7. Commit the Versant transaction.
8. Release the Versant session.

You do not need to run steps 3, 5, and 7 for calls that don't modify anything; such calls also do not take much execution time. If the incoming **RequirementService** call is **runActivator**, or **runScript**, then that Tcl runs within step 4. Hence, many Tcl calls and actions are carried out within the scope of one server request and one transaction.

You can execute many actions in one transaction by using the Internal Java API, specifically the JavaAPI class. It is invoked through the **RequirementService.tcrEvaluate** method. It crosses the **RequirementService** boundary once and runs the evaluate method of the passed **TcrExecutable** object within the transaction. From a transaction standpoint, it is similar to calling the **runActivator** or **runScript**, except that it runs Java instead of Tcl. The internal Java API is essential for optimal performance when accessing a large volume of fine grained data with multiple **getValue** or **getList** calls and/or following relations to visit multiple objects. If you are performing a single action, like Export or Import, then calling the **RequirementService** directly is acceptable.

To use the Internal Java API you must define your own class that implements the **TcrExecutable** interface, which means that it must have an evaluate method. Your evaluate method can call any of the JavaAPI class methods. You must write one class with an evaluate method for each unique task that you want Systems Architect/Requirements Management to accomplish from Java. This is quite similar to writing individual activators for specific tasks in Tcl.

When you want to run one of these unique tasks, you must create an instance of the class and pass it in a **RequirementService.tcrEvaluate** call. In this way, all JavaAPI calls of the evaluate method occur inside one transaction. In addition to avoiding the overhead of crossing the **RequirementService** boundary multiple times, these calls also benefit from the cache built up as you touch different objects.

If the JavaAPI calls are modifying objects, you have the safety of the entire action completing successfully and committing the transaction, or rolling back the entire transaction if it fails. While using the **RequirementService** API to perform a set of interrelated actions, you have the risk of a later step failing that could leave the incomplete results from the first steps in the database.

The decision to use the Internal Java API or the Tcl API for carrying out a specified task depends on the language that the developer is most comfortable with. The Internal Java API is always the most efficient choice. Regardless of whether the JavaAPI class or Tcl API is called, most of the execution time is the same for the Systems Architect/Requirements Management code under both APIs that are called to do the object-level work.

Both the JavaAPI and RequirementService classes are covered in the Systems Architect/Requirements Management JavaDoc.

Figure 3-1 is an example of an internal Java API.

```
package com.teamcenter.example;

import com.edsplm.tc.req.databeans.DataBean;
import com.edsplm.tc.req.enterprise.JavaApi;
import com.edsplm.tc.req.enterprise.TcrExecutable;
import com.edsplm.tc.req.enterprise.TcrObject;
import java.util.Vector;

/**
 * Internal Java API Example
 *
 */
public class InternalJavaAPIExample implements TcrExecutable
{

    public InternalJavaAPIExample()
    {
        super();
    }

    public void evaluate( JavaApi javaApi, Vector aVector ) throws Exception
    {
        TcrObject project = null;
        project = javaApi.createProject("ProjectOne");
        if(project == null)
        {
            return;
        }

        //Create a folder and requirement
        TcrObject folder = null;
        TcrObject requirement = null;
        try {
            folder = javaApi.createObject("Folder1", project,
DataBean.TYPE.FOLDER, "");
            requirement = javaApi.createObject("Requirement1", folder,

DataBean.TYPE.REQUIREMENT, "");
        } catch (Exception e) {
            e.printStackTrace();
        }

        //Rename the folder and requirement
        javaApi.setValue(folder, DataBean.PROPERTY.NAME, "InternalJavaAPIs");
        javaApi.setValue(requirement, DataBean.PROPERTY.NAME, "FirstTask");
    }
}
```

Figure 3-1. InternalJavaAPIExample.java

```
//package com.teamcenter.example;

import com.edsplm.tc.req.databeans.MessageBean;
import com.edsplm.tc.req.databeans.ResultBean;
import com.edsplm.tc.req.enterprise.RequirementService;
import com.edsplm.tc.req.database.dbutils.*;
import com.edsplm.tc.req.enterprise.Config;
import java.util.Iterator;
import java.util.Vector;
import java.util.HashMap;

/**
 * Class for testing Internal Java API
 *
 */
public class TestInternalJavaAPI
{

    public static void main(String args[])
    {
        HashMap argMap = DefineSystem.parseArguments(args);
        String dbName = (String) argMap.get("-db");
        if ((dbName == null) || (dbName.equals(""))) {
            dbName = "TCR_db"; // deafult value
        }

        DBManager dbm = new DBManager(dbName);
        System.setProperty(Config.DBNAME, dbName);

        InternalJavaAPIExample example = new InternalJavaAPIExample();
        Vector tcrObjects = new Vector();

 // As with all RequirementService calls there must be an active session for
”username”
//when making the tcrEvaluate call. Ordinarily that is done once at the beginning
of a
//user’s session.
// It is not necessary to have a separate authenticateUser call for each
tcrEvaluate call.

        ResultBean loginResult =
RequirementService.authenticateUser("username","passwd","");
        ResultBean result =
RequirementService.tcrEvaluate("username",tcrObjects,example);
        if (! result.isSuccess())
        {
            // Get error message
            Iterator i = result.getMessageList().iterator();
            while (i.hasNext())
            {
                MessageBean msg = (MessageBean)i.next();
                System.out.println("Failed with error: " + msg.getTag());
            }
```

```
            }
            else
            {
                System.out.print("Successfully executed test");
            }
        }

    }
```

Figure 3-2. TestInternalJavaAPI.java

Describing DataBeans

Systems Architect/Requirements Management objects are represented by **DataBeans**. A **DataBean** contains a table of the property values for the Systems Architect/Requirements Management object. The **DataBean** class also contains the constant values used in many API calls. When referring to these values, it is important to use the constants, instead of the actual values.

Java API Methods

The list of Java API functions is provided in the Javadocs. The Javadocs describe each function along with the response expected from the server.

Java API Examples

Figures 3-3, 3-4, and 3-5 work together to provide an example of selecting a project, a folder in the project, and creating a requirement in that folder. Figure 3-3 displays a drop down list of project names and allows you to select one.

```
<!-- This page is used for creating a requirement -->
<!-- This is the first page to be displayed -->
<!-- import statements -->
<%@ page import ="java.io.*"  %>
<%@ page import ="java.text.*"  %>
<%@ page import ="java.util.*"  %>
<%@ page import ="javax.servlet.*"  %>
<%@ page import ="javax.servlet.http.*"  %>

<%@ page import ="com.edsplm.tc.req.enterprise.*"  %>
<%@ page import ="com.edsplm.tc.req.databeans.*"  %>
<!-- Select a project for the list box -->
<HTML>
  <HEAD>
             <H1><B>Select Project</B></H1>
  </HEAD>
<BODY>
<!-- select Folder is the second page to be displayed -->
<FORM METHOD=POST ACTION="/tcr/custom/selectFolder.jsp" >
<%
             HttpSession aSession = request.getSession();
/* if the user is not logged-in, the log-in page will be displayed */
if ( aSession.getAttribute( "loggedIn" ) == null ) {
             response.sendRedirect( "/tcr/custom/login.jsp?originalURL=" +
             request.getRequestURI() );
}
else {
             String aUserName = (String) aSession.getAttribute( "userplusid" );
%>
<%
             /*
             * Returns the list of objects the user has access to and a specified
             * set of properties for each object on the list.
             *  To Gets the list of all projects in the database and their Name
             *  property <BR> <CODE>
             *  getList(userId,"",DataBean.LIST.PROJECT,new String[]
             *  {DataBean.PROPERTY.LOID}); </CODE> <BR>.
             */
             ResultBean resPro = RequirementService.getList(aUserName, "",
                                    DataBean.LIST.PROJECT, new String[]
                {DataBean.PROPERTY.LOID}); %>
             <% if (resPro.isSuccess() == false) { %>
                         <!-- If no project is found  -->
                         <br> Did not find any projects.
                         Successfully created the object= <%= resPro.isSuccess()
%>
             <% } %>
             <% /* Found at least one project */

                         Vector dbProject = (Vector)resPro.getResult();
                         Iterator idbProj = dbProject.iterator();
              %>
```

```
                Select a project from the list.
                <br>
```

Figure 3-3. SelectProject.jsp (Continued)

```
                <Select Name="project" SIZE = 1 onChange="form.submit()">
                <option VALUE = ""> </option>
                <%
                            /*        Gets the list of projects
                                       select the LOID and Name
                                       Populate the drop down list box
                            */
                            while (idbProj.hasNext()) {
                                       DataBean dbProj = (DataBean) idbProj.next();
                                                  %>
                                       <option VALUE = <%= dbProj.getObjectId()%>>
                                       <%= dbProj.getName() %></option>
                            <% }
                            }%>
                            </Select>
                            <br>

   </FORM>
   </BODY>
   </HTML>
```

Figure 3-3. SelectProject.jsp

Figure 3-4 displays a selectable list of folders from the project specified in figure 3-3. It also opens an editable text area to enter a requirements text.

```
<!-- This is the second page to be displayed to Creates a requirement -->
<!-- import statements -->
<%@ page import ="java.io.*"  %>
<%@ page import ="java.text.*"  %>
<%@ page import ="java.util.*"  %>
<%@ page import ="javax.servlet.*"  %>
<%@ page import ="javax.servlet.http.*"  %>

<%@ page import ="com.edsplm.tc.req.enterprise.*"  %>
<%@ page import ="com.edsplm.tc.req.databeans.*"  %>

<HTML>
  <HEAD>
             <H1><B>Select Folder</B></H1>
  </HEAD>
<BODY>
<!-- createObject will be the next page to be displayed -->
<FORM METHOD=POST ACTION="/tcr/custom/createObject.jsp" >
<%
             HttpSession aSession = request.getSession();
/* if the user is not logged-in, the login page will be displayed  */
if ( aSession.getAttribute( "loggedIn" ) == null ) {
             response.sendRedirect( "/tcr/custom/login.jsp?originalURL=" +
request.getRequestURI() );
}
else {
             String aUserName = (String) aSession.getAttribute( "userplusid" );
%>
<%  /*             This Enumeration is not required, To be removed in future */
             Enumeration enum = request.getParameterNames();
 // System.out.println( "enum:  " + enum.hasMoreElements() );
             while (enum.hasMoreElements()) {
```

Figure 3-4. SelectFolder.jsp (Continued)

```
                String inputName = (String)enum.nextElement();%>
                <%String inputValue = request.getParameter(inputName);%>
              <% } %>
<%
//  System.out.println("Project = " + request.getParameter("project"));
//   System.out.println("UserName = " + aUserName);
             /*
              * Returns the list of objects the user has access to and a
specified set
              * of properties for each object on the list.
              * @param aUserName
              * @param request.getParameter("project") LOID representing the
object
              * @param  DataBean.LIST.FOLDER name of the list. The list constants
              * are defined in DataBean.LIST class
              * @param new String[]{DataBean.PROPERTY.LOID} List of properties to
collect
              * for each object in the list.The DataBean
              * for each object will have the set of Properties defined in this
array.
              * The property constantsare defined in DataBean.PROPERTY
              * Vector of DataBean @Returns ResultBean The result data member of
the
              * ResultBean will have a  objects.
              */
              ResultBean resPro = RequirementService.getList(aUserName,
                           request.getParameter("project"),
                           DataBean.LIST.FOLDER, new
String[]{DataBean.PROPERTY.LOID}); %>
               <!-- if no folder is found resPro.isSuccess() will be false -->
               <% if (resPro.isSuccess() == false) { %>
                         <br>             Did not find any projects.
                           Successfully created the object= <%=
resPro.isSuccess() %>
              <% } %>

              <% Vector dbProject = (Vector)resPro.getResult();
                         Iterator idbProj = dbProject.iterator();
              %>
              Select a folder from the list.
              <br>
              <Select Name="folder" SIZE = 1 >
              <option VALUE = ""> </option>
              <%
                         /* Populate the drop down list box for the folders */
                         while (idbProj.hasNext()) {
                                    DataBean dbProj = (DataBean) idbProj.next();
                                               %>
                                       <option VALUE = <%=
dbProj.getObjectId()%>>
<%= dbProj.getName() %></option>
                         <% } %>
                         <% } %>
```

```
                          </Select>
                          <br>


<!-- Creates a text area for the input requirement -->

</TABLE><p>Enter Text</p>

<p> <textarea wrap=virtual name=bodyText cols=50 rows=10>
             </textarea>
<!-- Creates action buttons  -->
<input type=submit value=Ok>
<input type=reset value=Reset>
<input type=button value=Cancel onClick=self.close() >
</FORM>
</BODY>
</HTML>
```

Figure 3-4. SelectFolder.jsp

Figure 3-5 creates a new requirement in the folder specified in figure 3-4 and sets the text to what was entered in above.

```
<!-- This is the 3rd and final page to be displayed when a requirement is created
-->
<!-- import statements -->
<%@ page import ="java.io.*"  %>
<%@ page import ="java.text.*"  %>
<%@ page import ="java.util.*"  %>
<%@ page import ="javax.servlet.*"  %>
<%@ page import ="javax.servlet.http.*"  %>

<%@ page import ="com.edsplm.tc.req.enterprise.*"  %>
<%@ page import ="com.edsplm.tc.req.databeans.*"  %>

<HTML>
  <HEAD>
  </HEAD>
<BODY>
<FORM METHOD=POST  >
<%
             HttpSession aSession = request.getSession();
/* If the user is not logged-in, the login page will be displayed  */
if ( aSession.getAttribute( "loggedIn" ) == null ) {
             response.sendRedirect( "/tcr/custom/login.jsp?originalURL=" +
             request.getRequestURI() );
}
else {
             String aUserName = (String) aSession.getAttribute( "userplusid" );
%>
<%

%>
<%  Enumeration enum = request.getParameterNames();
             while (enum.hasMoreElements()) {
               String inputName = (String)enum.nextElement();%>
               <%String inputValue = request.getParameter(inputName);%>
             <% } %>

<%
              /**
              * Creates a new object in the database
              * @param aUserName The User Session ID of the user
              * @param request.getParameter("folder") the LOID of the owner
object
              * if the value of the position parameter
              * sibling (DataBean.POSITION) is FIRST_MEMBER or LAST_MEMBER;
              * or the LOID of the object if the value of the position parameter
is
              * NEXT_SIBLING or LAST_SIBLING.
              * @param "Requirement" The type of object to Createsin the
database.
              * This could be any of the constant type names defined in
DataBean.TYPE.
              * or the name of the UserTypeDefinition object
              * @param DataBean.POSITION.LAST_MEMBER Keyword specifying the
```

```
   location
                * of the new object relative to ownerId. See DataBean.POSITION
                * for more details @Returns ResultBean. The result data member of
   the
                * ResultBean will actually have
                * the DataBean of the object created.
```

Figure 3-5. CreateObject.jsp (Continued)

```
examples:
                        To Creates a Child Requirement on owner id (ex.
123.45.6789)
                        <CODE><BR>

createObject(userId,123.45.6789,DataBean.TYPE.REQUIREMENT,
                        DataBean.POSITION.LAST_MEMBER);
                        </CODE>
                        To Creates a next sibling Requirement on sibling
123.45.6789
                        <CODE><BR>

createObject(userId,123.45.6789,DataBean.TYPE.REQUIREMENT,
                        DataBean.POSITION.NEXT_SIBLING);
                        </CODE>
             */
              ResultBean res = RequirementService.createObject(aUserName,
              request.getParameter("folder"), "Requirement", DataBean.POSITION.
              LAST_MEMBER  );
%>

<% /* if the object was not created res.isSuccess() will be false */
             if (res.isSuccess() == false) { %>
<br>             The object was not created.
              Successfully created the object= <%= res.isSuccess() %>
              <br>
<% } else { %>
<%  DataBean db = (DataBean)res.getResult(); %>
<%
             Set mySet = db.getPropertyKeySet();
             Iterator itr = mySet.iterator();
%>

<% /* set the text on the newly created requirement */
             ResultBean rb = RequirementService.setObject(aUserName,
db.getObjectId(),
              new String[]{"Text"},
             new String[]{request.getParameter("bodyText")}, 0, ""); %>
             <!-- Display if the requirement text update was successful or not --
>
              It was a success =<%= rb.isSuccess() %>
              <br>
              <!-- Creates an action button -->
              <input type=button value=Close onClick=self.close() >

<% } %>



</FORM>
</BODY>
```

```
</HTML>
<% } %>
```

Figure 3-5. CreateObject.jsp

Figures 3-6 and 3-7 provide an example of searching a project for objects that match specified search criteria.

Figure 3-6 displays a selectable list of projects and search options. You can select the object types, object name, text, and whether the search is case sensitive.

```
<%@ page import ="java.io.*"  %>
<%@ page import ="java.text.*"  %>
<%@ page import ="java.util.*"  %>
<%@ page import ="javax.servlet.*"  %>
<%@ page import ="javax.servlet.http.*"  %>
<%@ page import ="com.edsplm.tc.req.enterprise.*"  %>
<%@ page import ="com.edsplm.tc.req.databeans.*"  %>

<!--
  This is the first page to be display for Text search.
  Next jsp pages to be activated is searchObject.jsp
-->
<FORM METHOD=POST ACTION="/tcr/custom/searchObject.jsp" >
<%
/*
If the user is not logged-in, the login form is presented.
*/
HttpSession aSession = request.getSession();
if ( aSession.getAttribute( "loggedIn" ) == null ) {
             response.sendRedirect( "/tcr/custom/login.jsp?originalURL=" +
             request.getRequestURI() );
}
else {
             String aUserName = (String) aSession.getAttribute( "userplusid" );
%>
<H1> Search for Object:  </H1>
<%
              ResultBean resPro = RequirementService.getList(aUserName, "",
              DataBean.LIST.PROJECT, new String[]{DataBean.PROPERTY.LOID}); %>



<!-- ResultBean.isSuccess == true, at least a project was found
                           ResultBean.isSuccess == false, no projects were found
               -->
               <% if (resPro.isSuccess() == false) { %>
                         <br>             Did not find any projects.
                           Successfully created the object= <%=
resPro.isSuccess() %>
              <% } %>
              <!--
              Iterate through getResult of the ResultBean
              -->
              <% Vector dbProject = (Vector)resPro.getResult();
                         Iterator idbProj = dbProject.iterator();
              %>
              Select a project.
              <br>
              <!-- Provide a blank selection -->
              <Select Name="StartObj" SIZE = 1 >
              <option VALUE = ""> </option>
<!-- Populate the drop down list box with the list of projects -->
              <%
```

```
                                        while (idbProj.hasNext()) {
                                    DataBean dbProj = (DataBean) idbProj.next();
```

Figure 3-6. Search.jsp (Continued)

```
                                                    %>
                                               <option VALUE = <%=
dbProj.getObjectId()%>>

                          <%= dbProj.getName() %></option>
                           <% } }%>
                           </Select>
                           <br>


<%--
<P>Project ID or Folder where search should start<br><INPUT TYPE=text
NAME=StartObj
SIZE=30>
--%>
<br><br>
<P>Name<br><INPUT TYPE=text NAME=Name SIZE=30>
<br><br>
Content(Requirement & Note Only)<br><INPUT TYPE=text NAME=Content
SIZE=30></P>
<br>
<!-- Createsthe check boxes for Folder, Requirement and Note -->
<P><table>
             <tr>
                          <td>
                                       <INPUT TYPE=checkbox NAME="Folder"
VALUE="Yes">
                          </td>
                          <td>
                                       Folder
                          </td>
             </tr>
             <tr>
                          <td>
                                       <INPUT TYPE=checkbox NAME="Requirement"
                                       VALUE="Yes" checked>
                          </td>
                          <td>
                                       Requirement
                          </td>
             </tr>
             <tr>
                          <td>
                                       <INPUT TYPE=checkbox NAME="Note" VALUE="Yes">
                          </td>
                          <td>
                                       Note
                          </td>
             </tr>
             <tr><td><br></td></tr>
      <!-- Check box for the case sensitive search -->
      <tr>
             <td>
```

```
                    <INPUT TYPE=checkbox NAME="caseSensitive" VALUE="Yes">
                </td>
                <td>
                    Case Sensitive
                </td>
        </tr>
    </table></P>
    <!-- create the action buttons -->
    <input type=submit value=Ok>
    <input type=reset value=Reset>
    <input type=button value=Cancel onClick=self.close() >
```

Figure 3-6. Search.jsp

Figure 3-7 performs the search and displays the results.

```
<!-- import statements -->
<%@ page import ="java.io.*"  %>
<%@ page import ="java.text.*"  %>
<%@ page import ="java.util.*"  %>
<%@ page import ="javax.servlet.*"  %>
<%@ page import ="javax.servlet.http.*"  %>

<%@ page import ="com.edsplm.tc.req.enterprise.*"  %>
<%@ page import ="com.edsplm.tc.req.databeans.*"  %>

<%
 int x = 0;
 int k = 0;
 boolean caseSen =  false;

HttpSession aSession = request.getSession();
/* if the user is not logged-in provide the login form */
if ( aSession.getAttribute( "loggedIn" ) == null ) {
             response.sendRedirect( "/tcr/custom/login.jsp?originalURL=" +
request.getRequestURI() );
}
else {
             String aUserName = (String) aSession.getAttribute( "userplusid" );
/* iterate through the request to Gets the list of the listed check boxes */
 Enumeration enum = request.getParameterNames();
 while (enum.hasMoreElements()) {
             String inputName= (String)enum.nextElement();%>
             <%String inputValue = request.getParameter(inputName);%>
             <%}
%>

<!-- count the number of check boxes selected and assign this value to x -->
<% if (request.getParameter("Folder") != null && request.getParameter("Folder")
.equals("Yes")) {
             x = x+1; }
   if (request.getParameter("Requirement") != null && request.getParameter("Requirement")
.equals("Yes"))
             {x = x+1; }
   if (request.getParameter("Note") != null && request.getParameter("Note") .equals("Yes"))
{
             x = x+1; }
             String[] SelectFrom = new String[x];
             /*select a an array of objects to select from. It could have Folder,
Requirement and Note.*/
   if (request.getParameter("Folder") != null && request.getParameter("Folder")
.equals("Yes")) {
             SelectFrom[k] = DataBean.TYPE.FOLDER;
             k = k+1;}%>

   <%if (request.getParameter("Requirement") != null && request.getParameter("Requirement")
.equals("Yes")
             ) {SelectFrom[k] = DataBean.TYPE.REQUIREMENT;
             k = k+1;}
```

```
   if (request.getParameter("Note") != null && request.getParameter("Note") .equals("Yes"))
{
             SelectFrom[k] = DataBean.TYPE.NOTE;}
             /* If the search has to be case sensitive, assign true to caseSen. */
   if (request.getParameter("caseSensitive") != null &&
request.getParameter("caseSensitive")
             .equals("Yes"))
             caseSen = true;
%>
<% /*Search the selected object types for the text */
             for(int i= 0;  i <SelectFrom.length; i++)
             System.out.println(SelectFrom[i]); %>

<%  /*
              * Search the database for objects that match the specified criteria.
              * @param userName The sessionID + userName
              * @param request.getParameter("StartObj") The project or folder object ID
              * where search should start
              * @param request.getParameter("Name") The object name search criterion
              * @param request.getParameter("Content") The requirements content search
criterion
              * @param SelectFrom Objects types to search for (RequirmentDB, NoteDB,
FolderDB)
              * @param new String[]{"Text"} The desired properties to Returns for each
object
              * @param caseSen if true the search is case sensitive
              * @Returns Collection of DataBeans that match search criteria
              */
             ResultBean rb = RequirementService.search(aUserName,
request.getParameter("StartObj"),
             request.getParameter("Name"), request.getParameter("Content"), SelectFrom, new
String[]
             {"Text"} , caseSen);%>

                          <% if (rb.isSuccess() == false) { %>
                          <!-- Error in finding selected object -->
                          <br>The text was entered successfully = <%= rb.isSuccess() %>
                          <br>
                          <input type=button value=Close onClick=self.close() >

             <% } else {%>
                          <%
                          /* No error, Match may or may not be found */
                          if (rb.isSuccess() == true ) {
                          Vector vdb = (Vector)rb.getResult();
                          Iterator itr = vdb.iterator();
                          %>
```

Figure 3-7. SearchObject.jsp (Continued)

```
                                <H2> Found : <%= vdb.size() %>  </H2>
                                <TABLE BORDER=2>
                                <TR>
                                <!-- Column headings -->
                                                                <TD><b><i>Name </b> </i></TD>
                                <TD><b><i>Type</i></b></TD>
                                                                 <TD><b><i>Text</i></b></TD>
                                                                 <TD><b><i>Send email</i></b>


        </TD>
                        </TR>
                                <%
                                while (itr.hasNext()) {
                                            DataBean db = (DataBean) itr.next();
                                %>
                                <TR>
                                            <!-- Createslink to the text object -->
                                            <TD><B><% if (db.getType().equals
                                            (DataBean.TYPE.REQUIREMENT)) {%>
                                <a href="/tcr/custom/getText.jsp?loid=
                                            <%= db.getObjectId() %>
                                " ><%= db.getName() %></a>
                        <% } else { %>
                                <%= db.getObjectId()%>
                        <% } %>
                                </B></TD>

                        <TD>
                                <%= db.getType() %>
                                </TD>

                                 <TD>
                                 <%
                                 /**
                                 * Gets the given object from the database with set
                                 * of properties
                                 * @param aUserName User Session id of the user
                                 * @param db.getObjectId() LOID of the object
                                 * @param new String[]{DataBean.PROPERTY.HTML}
                                 * List of properties to return.
                                 * Keyword can include
                                 * DataBean.ALL_PROPERTY/SYSTEM_PROPERTY/USER_PROPERTY
                                 * @param openForEdit True if the user is opening
                                 * the object with the
                                 * intent to modify the object.  This will place
                                 * a reservation (lock) on the object
                                 * to prevent others from modifying the object.
                                 * The caller must use the
                                 * RequirementService.releaseObject method to
                                 * clear the reservation.
                                 * This flag is used only for Requirement and Note
                                 * objects and DataBean.PROPERTY.MHTML is in
```

```
                                * desiredProps.
                                       *  For all other object types and property
value,
                                * this flag is ignored.
                                * @Returns ResultBean The result data member of
                                * the ResultBean
                                * will have the DataBean for
                                * the object and its desiredProps. The schemaList
                                * data member
                                * of the ResultBean will have
                                * a data bean for Property Definitions
                                * for each of the
                                * property names in desiredProps. This
                                * gives the caller all the necessary information
                                * to edit the properties of this object.
                                *
                                */
                                ResultBean res = RequirementService.getObject
                                (aUserName, db.getObjectId(),new
                                String[]{DataBean.PROPERTY.HTML},
                                false);
                                DataBean obj = (DataBean)res.getResult();
                                %>
                                <%= obj.get(DataBean.PROPERTY.HTML)   %>
                                </TD>
                                <TD>
                                <a href="mailto:?subject=Approval is
                                requested!&body=/tcr/custom/getText.jsp?
                                loid=<%= db.getObjectId()%> Please click on the link
                                to read the text.">
                                Click here</a> to send an email for approval.
```

Figure 3-7. SearchObject.jsp (Continued)

```
            </TD>
</TR>
            <% } %>
<!-- close button -->
</TABLE><br><input type=button value=Close
onClick=self.close() >
<% } %>
<% } %>
                              <% } %>
```

Figure 3-7. SearchObject.jsp

Figure 3-8 provides an example of using the Teamcenter SSO for validation. The is **loginUsingTcSS.jsp** file is available in the **custom** directory of the **tcr.war** file.

```
<!-- This page is presented to the user when the user is not logged-in -->
<!-- Import Statements -->
<%@ page import="java.io.*,java.util.*" %>
<%@ page import="com.edsplm.tc.req.web.utils.ParamWebXml" %>
<%@ page import ="java.text.*"  %>
<%@ page import ="javax.servlet.*"  %>
<%@ page import ="javax.servlet.http.*"  %>
<%@ page import="com.edsplm.tc.req.database.API" %>
<%
String warFileName = API.getWarFileName();
%>

<html>
<head>
<title>Teamcenter 10 for systems engineering</title>
</head>
<body bgcolor="#ffffff"  topmargin="10" leftmargin="10" MARGINWIDTH="10" MARGINHEIGHT="10"
>
<jsp:useBean id="validationBean" scope="page"
class="com.edsplm.tc.req.web.utils.CrossSiteValidationBean"/>
<script language= JavaScript>
</script>

<!-- Checking the authorization Error by getting the attribute value for
"TcR.AuthenticateError"
     and checking for cross site validation -->
<%
String authError = (String) request.getSession().getAttribute("TcR.AuthenticateError");
if(!validationBean.isInputValidAgainstCrossSiteScripting(authError)) {
    response.sendRedirect("/" + warFileName + "/ugs/tc/req/CrossSiteScriptingThreat.jsp");
    return;
}

// Setting response header and ssoEnabled
response.setHeader("TcR.AuthenticateRequired", "TcR.AuthenticateRequired");
boolean ssoEnabled = false;

// Getting the ParamWeb value for SSO.Enabled
try {
    String temp = ParamWebXml.getConfigParam("SSO.Enabled",
getServletConfig().getServletContext());
    if (temp.equalsIgnoreCase("true")) {
            ssoEnabled = true;
    }
} catch (Exception e) {
    e.printStackTrace();
}

// If the SSO is enabled, authorization error is checked
if (ssoEnabled) {
    if (authError != null && authError.length() > 0) {
            <!-- Architect/Requirement is unable to authenticate user after SSO login -->
```

```
%>

<body bgcolor="#ffffff"  topmargin="0" leftmargin="0" MARGINWIDTH="0" MARGINHEIGHT="0" >

<jsp:include page="copyright_table.html"/>
<!-- Restrict table to 750 pixels so that it fits in 800x600 popup login screen -->
<table border="0" cellspacing="0" cellpadding="0" width="750" align="center">
    <tr>
           <td> </td>
    </tr>
    <tr>
           <td align="left" width="35%" valign="top" ></td>
           <td width="65%" align="right">
```

Figure 3-8. loginUsingTcSS.jsp (Continued)

```
                    <table cellspacing="0" cellpadding="8" border="0">
                            <tr>
                                    <td><p><font color=red><b>Unable to authenticate user in
Teamcenter 10 for Systems Engineering.
                                    <p>Error: <%= authError %>.
                                    <p>Please contact your Teamcenter administrator</b></td>
                                    <td></td>
                            </tr>
                    </table>
            </td>
    </tr>
</table>
</body>
</html>

<%
    // ssoEnabled continued
            out.close();
            return;
    }

    // Get the SSO login URL
    String ssoLoginURL = ParamWebXml.getConfigParam("SSO.LoginURL",
getServletConfig().getServletContext());

    // If the SSOLogin URL is null, throw exception
    if (ssoLoginURL == null) throw new Exception("SSO.LoginURL param missing from
web.xml");

    // Else append the SSO login URL
    ssoLoginURL += "/weblogin/login_redirect";

    // Get the SSO AppID
    String ssoAppID = ParamWebXml.getConfigParam("SSO.AppID",
getServletConfig().getServletContext());
    if (ssoAppID == null) throw new Exception("SSO.AppID param missing from web.xml");

    // Get the URL TcR.LoginRequestURL
    String url = (String) request.getSession().getAttribute("TcR.LoginRequestURL");
    if(!validationBean.isInputValidAgainstCrossSiteScripting(url) ){
        response.sendRedirect("/" + warFileName +
"/ugs/tc/req/CrossSiteScriptingThreat.jsp");
        return;
    }

    int uriStart = -1;

    // Check for URL
    if( url != null){
            uriStart = url.indexOf("/tcr");
    }
```

```
    // If URL is null, throw an exception
    if (uriStart < 0) throw new Exception("Unable to determine SSO return URI. Request URL
is: " + url);
    String ssoURI = url.substring(uriStart, url.length());

    // Create a form and auto-post it to the SSO login URL
    String launch_mode = request.getParameter("launch_mode");
    if(launch_mode == null){
        launch_mode = "";
    }
    if(!validationBean.isInputValidAgainstCrossSiteScripting(launch_mode) ){
        response.sendRedirect("/" + warFileName +
"/ugs/tc/req/CrossSiteScriptingThreat.jsp");
        return;
    }

    String html = "<body onLoad=\"document['redirect'].submit()\" >"
           + "<form name=\"redirect\" action=\"" + ssoLoginURL + "\" method=POST >"
           + "<input type=\"hidden\" name=\"TCSSOAPPID\" value=\"" + ssoAppID + "\">"
           + "<input type=\"hidden\" name=\"TCSSORURI\" value=\"" + ssoURI + "\">"
           + "<input type=\"hidden\" name=\"launch_mode\" value=\"" + launch_mode + "\">";

// Pass the request parameters
    java.util.Enumeration paramNames = request.getParameterNames();
    while (paramNames.hasMoreElements()) {
           String paramName = (String) paramNames.nextElement();
           String paramValue = request.getParameter(paramName);
        if(!validationBean.isInputValidAgainstCrossSiteScripting(paramName) ||
                !validationBean.isInputValidAgainstCrossSiteScripting(paramValue)) {
            response.sendRedirect("/" + warFileName +
"/ugs/tc/req/CrossSiteScriptingThreat.jsp");
            return;
        }
           html += "<input type=\"hidden\" name=\"" + paramName + "\" value=\"" +
paramValue + "\">";
    }

    html += "</form></body>";

    out.println(html);
    out.close();
    return;
}

%>
```

Figure 3-8. loginUsingTcSS.jsp (Continued)

```
<script language= JavaScript>

   function writeCookie() {
           var nextyear = new Date();
           var user = document.login.j_username.value;
           if (user == "") {
                   alert("User Name field must not be blank");
                   return false;
           }
        <!-- Encode the username and password -->
        var password = encodeURIComponent(document.login.j_password.value);
           document.login.j_password.value = password;
        document.login.j_username.value = encodeURIComponent(user);
           nextyear.setFullYear(nextyear.getFullYear()+1);
           document.cookie = "TcRLogin="+ escape(user) + ";
path=/;expires="+nextyear.toGMTString();
           return true;
    }
    function readCookie() {
           var allcookies = document.cookie;
           var pos = allcookies.indexOf("TcRLogin=");
           var value = "";
           if (pos != -1) {
                   var start = pos + 9;
                   var end = allcookies.indexOf(";", start);
                   if (end == -1) end = allcookies.length;
                   var value = allcookies.substring(start, end);
                   value = unescape(value);
           }
           document.login.j_username.value = value
    }

    function setFocus() {
        if (document.forms[0].j_username.value.length > 0) {
        }
           else {
        }
    }

    function cancelLogin (queryParams)
    {
        var url = '/<%=warFileName%>/ugs/tc/req/CloseApplication.jsp';
        if (queryParams != null && queryParams.length > 0)
        {
           url += ('?' + queryParams);
        }

        location.href = url;
    }
</script>
```

Figure 3-8. loginUsingTcSS.jsp (Continued)

```
<!-- The following page is displayed when the user is already logged on -->
<!-- Architect/Requirements Login Page -->
<body bgcolor="#ffffff"  topmargin="0" leftmargin="0" MARGINWIDTH="0" MARGINHEIGHT="0">

<jsp:include page="copyright_table.html"/>

<table border="0" cellspacing="0" cellpadding="0" width="750" align="center">

    <form name="login" method="POST" action="/<%=warFileName%>/custom/loggingIn.jsp" >

    <tr>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
    </tr>

    <tr>
            <td>    </td>
            <td align="left" width="35%" valign="top" ></td>
            <td width="65%" align="right">
                    <table cellspacing="0" cellpadding="8" border="0">
                            <tr>
                                    <%
                                    if (authError != null && authError.length() > 0) {
                                out.write("<td><font color=red><b>" + authError +
"</b></font></td>");
                                 }
                                    String invalidLicense = (String)
session.getAttribute("TcR.InvalidLicense");
                                    String invalidPassword = (String)
session.getAttribute("TcR.InvalidPassword");

if(!validationBean.isInputValidAgainstCrossSiteScripting(invalidLicense) ||

!validationBean.isInputValidAgainstCrossSiteScripting(invalidPassword)) {
                    response.sendRedirect("/" + warFileName +
"/ugs/tc/req/CrossSiteScriptingThreat.jsp");
                       return;
                    }
                                    %>
                                    <script language=JavaScript>
                                            var invalidPassword = "<%= invalidPassword %>"
                                            if ( invalidPassword == "true" ) {
                                                    document.write("<td><font
color=red><b>Invalid username and/or password.</b></font></td>");
                                            }
                                            var licenseThere = "<%= invalidLicense %>"
                                            if ( licenseThere == "true" ) {
                                                    document.write( "<td><font
color=red><b>Invalid server license.  Contact your TCR administrator.
                                            </b></font></td>" );
                                            }
```

```
                                </script>
                            </tr>
                            <tr>
                                <td><nobr><strong><font face="Arial" size=2>User
Name:</font></strong></nobr></td>
                                <td><input style="TCInput" type=text id="userID"
name="j_username" size="25"></td>
                            </tr>
                            <tr>
                                <td><strong><font face="Arial"
size=2>Password:</font></strong></td>
                                <td><input style="TCInput" type=password
name="j_password"  value="" size="25"></td>
                            </tr>
                            <tr>
                                <td valign=middle><b><font face="Arial" size=2>Language:
</font></b></td>
                                <td valign=middle><SELECT name='LOCALE_PARAM'>
    <OPTION value="en_US" SELECTED>English (United States) - Default</OPTION>

</SELECT></td>
                            </tr>
                            <tr>
                                <td valign=top colspan=2 align=right><input
align="middle" type="Submit" value="   Log In   ">
                <%
                if (authError != null && authError.length() > 0) {

                    StringBuffer url = request.getRequestURL();

                    String queryStr = request.getQueryString();

if(!validationBean.isInputValidAgainstCrossSiteScripting(url.toString()) ||

!validationBean.isInputValidAgainstCrossSiteScripting(queryStr)) {
                    response.sendRedirect("/" + warFileName +
"/ugs/tc/req/CrossSiteScriptingThreat.jsp");
                    return;
                 }
                                                    if (url != null &&
url.lastIndexOf("interface_login") != -1) {
                        %>
                        <input type="button" value=" Cancel Login "
                               align="center"
                               onclick="javascript:cancelLogin ('<%= queryStr %>')" />
                        <%
                    }
                }
                %>
                    </td>
                            </tr>
                       </table>
            </td>
```

```
        </tr>
```

Figure 3-8. loginUsingTcSS.jsp (Continued)

```
            <%
            // Pass all the request parameters
            java.util.Enumeration paramNames = request.getParameterNames();
            while (paramNames.hasMoreElements()) {
                   String paramName = (String) paramNames.nextElement();
                   String paramValue = request.getParameter(paramName);
             if(!validationBean.isInputValidAgainstCrossSiteScripting(paramName) ||
                    !validationBean.isInputValidAgainstCrossSiteScripting(paramValue)) {
                response.sendRedirect("/" + warFileName +
"/ugs/tc/req/CrossSiteScriptingThreat.jsp");
             }
            %>
                   <input type="hidden" name="<%= paramName %>" value="<%= paramValue %>" >
            <%
            } // end while
            %>

    </form>
</table>

</body>
</html>
```

Figure 3-8. loginUsingTcSS.jsp

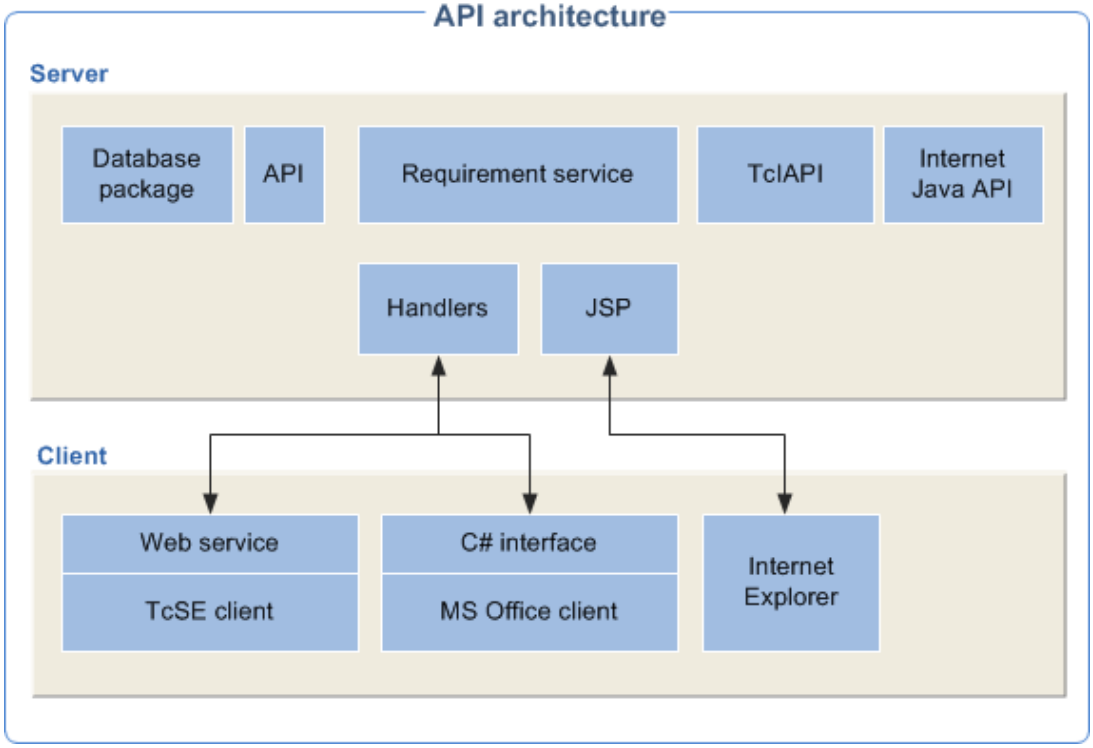
Chapter 4: Using C# API from COM (VBA)

This chapter contains information about using the C# API from COM (Component Object Model).

Introduction

The C# API methods provide functionality to make server calls from a COM client such as the VBA (Visual Basic for Applications) macro.

The following figure provides a graphical representation of the architecture of the Systems Architect/Requirements Management API.



The C# API contains methods to make Systems Architect/Requirements Management server calls. Interfaces are provided to make the calls using .NET, VBA or COM. The API methods are contained in

TcR.dll and **TcR.tlb** files. The **TcR.tlb** contains the information required to use procedures or classes in the **TcR.dll** file. The important components of API include:

ITcRConnection — **ITcRConnection** interface provides methods for making server calls from .NET code using VBA or COM.

ITcSEConnection — **ITcSEConnection** interface provides methods for making API calls using .NET and exposes the API methods to COM. **ITcSEConnection** is a wrapper around **ITcRConnection** to support COM-specific parameter format.

ResultBean — The API methods return a **ResultBean**. In addition to the requested data, the **ResultBean** contains status information about the call. This includes a flag indicating if the call succeeded and any error, warning or informational messages generated during the call. The success flag must be checked after each API call and error messages can be extracted and displayed. If no error occurs, the **ResultBean** contains the desired return value for the API call, typically a **DataBean** or list of **DataBean** that represent Systems Architect/Requirements Management objects.

Systems Architect/Requirements Management uses beans to return data from the Systems Architect/Requirements Management server to the client applications. In Java, a bean is a type of object that meets certain criteria. This includes the ability to be passed from a web server to a client application.

DataBean — A **DataBean** represents a single Systems Architect/Requirements Management object. A **DataBean** contains the name, type and unique database identifier (LOID) of an object. **Databean** also contain a list of property names and values.

DataBean constants — There are a number of **DataBean** sub classes that contain keywords for use in API calls. For example, the **DataBean.LIST** class contains the legal values for the listType argument of the **getList** API method. When referring to these values, you must use these constants instead of the actual values.

For more information on on Systems Architect/Requirements Management's public API, see the online API manual.

Configuring COM Client (VBA)

This section specifies the prerequisites and describes the procedures to get started with programming in VBA, a COM client.

Prerequisites

- Systems Architect/Requirements Management Client must be installed.

Adding Reference to TcR.tlb

Adding reference to **TcR.tlb** enables a VBA macro to make calls to Systems Architect/Requirements Management C# API.

1. Open the Visual Basic Editor and choose **Tools→References**.
2. Browse to the location and select **TcR.tlb**.

Connecting to Systems Architect/Requirements Management

A connection to the Systems Architect/Requirements Management server is made using the **TcRConnectionService** class. You can connect to a Systems Architect/Requirements Management server through the Systems Architect/Requirements Management client or directly to the Systems Architect/Requirements Management server. **TcRConnectionService** first attempts to connect through the Systems Architect/Requirements Management client. If a Systems Architect/Requirements Management client is not running then a direct connection to the Systems Architect/Requirements Management server is used.

A connection through the Systems Architect/Requirements Management client uses the **TcRChannel** class. If the Systems Architect/Requirements Management client is not running or not configured (connection port is not specified), it connects to the Systems Architect/Requirements Management server directly using **TcRWebService**. You can use the These **TcRWebService** or the **TcRChannel** class instead of **TcRConnectionService** class if you need to control the type of connection.

Siemens PLM Software recommends using **TcRConnectionService** to establish the connection. **TcRConnectionService** automatically selects the appropriate connection type.

To make a connection, the following information is required.

- Machine name of the Systems Architect/Requirements Management server
- Port used by the application server
- Valid Systems Architect/Requirements Management user ID and password

When connecting through the Systems Architect/Requirements Management client, you also need to specify a port number for communicating with the Systems Architect/Requirements Management client.

- Connect through **TcRConnectionService**

  This is the preferred way to connect to Systems Architect/Requirements Management. It is used to obtain an instance of either **TcRChannel** or **TcRWebService**.

```
Dim connectionService As TcR.TcRConnectionService
Dim tcrConnection As TcR.ITcSEConnection
Set connectionService = New TcR.TcRConnectionService
connectionService.tcr_client_socket_port = "4000"
connectionService.tcr_client_socket_url = "tcseclient"
connectionService.tcr_server_controller_url = _

      "http://tcseserver:8080/tcr/controller/"
connectionService.user_name = "tcseuser"
connectionService.user_password = "password"

Set tcrConnection = connectionService.CONNECTIONCOM
     If (tcrConnection.connectCOM()) Then
        If (tcrConnection.isConnectedCOM = CONNECTION_CLIENT) Then
            MsgBox "Connection to TcRChannel Successful"
        ElseIf (tcrConnection.isConnectedCOM = CONNECTION_SERVER) Then
            MsgBox "Connection to TcRWebService Successful"
        End If
    Else
        MsgBox "Connection to TcR Failed"
    End If
```

  The following is the explanation of the parameters used in the above example:

Parameter	Definition
tcr_client_socket_port	The port on which the Systems Architect/Requirements Management client listens and the data is communicated. For example, **"4000"** Do not set if the Systems Architect/Requirements Management client is not installed. The connection happens automatically. Ports available for use are specified in the **PortRangeStart** and **PortRangeEnd** web application parameters.  A Microsoft Office application launched from the Systems Architect/Requirements Management client uses one of the ports making it unavailable.
tcr_client_socket_url	The machine name on which the Systems Architect/Requirements Management client is installed. For example, **"tcseclient"** Do not set if the Systems Architect/Requirements Management client is not installed. The connection happens automatically.
tcr_server_controller_url	The controller URL of the server. You must provide the URL of the server as a value to this parameter. For example, **"http://tcseserver:8080/tcr/controller/"**.
user_name	The Systems Architect/Requirements Management user name. For example, **"tcruser"**
user_password	The password for the login user name provided as value to the **user_name** parameter. For example, **"password"**

The above example indicates that the Systems Architect/Requirements Management client is installed from the **http://tcseserver:8080/tcr** server on the machine with name **tcseclient** and the client is running on the port **4000**.

You can determine the name of the machine dynamically (at the run time) using the following method:

```
Private Declare Function GetComputerName Lib "kernel32" _
    Alias "GetComputerNameA" _
    (ByVal lpBuffer As String, nSize As Long) As Long

Function ReturnComputerName() As String
Dim rString As String * 255, sLen As Long, tString As String
    tString = ""
    On Error Resume Next
    sLen = GetComputerName(rString, 255)
    sLen = InStr(1, rString, Chr(0))
    If sLen > 0 Then
        tString = Left(rString, sLen - 1)
    Else
        tString = rString
```

```
    End If
    On Error GoTo 0
    MsgBox UCase(Trim(tString))
    ReturnComputerName = UCase(Trim(tString))
End Function
```

- Connect through **TcRWebService**

  This method is used to directly communicate with the Web server using the **TcRWebService** instance. This is an HTTP communication.

```
Dim webService As TcR.TcRWebService
Dim tcrConnection As TcR.ITcSEConnection

Set webService = New TcRWebService
webService.TcRControllerCOM = _
                    "http://pni3p179:8080/tcr/controller/"
webService.user_idCOM = "test"
webService.passwordCOM = "password"
If (webService.connectCOM()) Then
     MsgBox "Connection to TcRWebService Successful"
 Else
     MsgBox "Connection to TcRWebService Failed"
End If
```

- Connect through **TcRChannel**

  This method is used to communicate through a socket with the Systems Architect/Requirements Management client. The Systems Architect/Requirements Management client then forwards the call to the Systems Architect/Requirements Management Web server. This is a socket service communication. As **TcRConnectionService** attempts to connect using **TcRChannel** first, avoid using **TcRChannel** directly to make a connection.

  When connecting through the Systems Architect/Requirements Management client any modifications to database objects is refreshed in the client.

You must always use the **TcRConnectionService** instance to get the right connection type (**TcRWebService** or **TcRChannel**).

VBA Examples for Calling C# API Methods

This section provides a few examples for using C# API methods from VBA. For a list of all the methods, visit the Systems Architect/Requirements Management client launch page and click **API C# Doc**.

A working example of using the API methods from VBA and C# is provided with the Systems Architect/Requirements Management release package. You must have Microsoft Visual Studio 2010 installed to view the example from C#.

For more information on the examples, see Appendix - Examples for using C# API's.

createObjectCOM

DESCRIPTION

Creates an object in TcR and sets the given properties. It is a wrapper around the **createObject** method.

SYNTAX

```
Public Function createObjectCOM ( _
   owner As String, _
   type As String, _
   position As String, _
   ByRef propertyNames As String(), _
   ByRef propertyValues As String() _
) As ResultBean
```

ARGUMENTS

- `owner (String)`: Object ID of the owner of the new object.
- `type (String)`: Object type defined in **DataBean.TYPE**.
- `position (String)`: Object position defined in **DataBean.POSITION**.
- `propertyNames (String ())`: String array for the property names.
- `propertyValues (String ())`: String array for the property values.

RETURNS

A **ResultBean** object from Systems Architect/Requirements Management.

EXAMPLE

```
‘Define variables
Dim propName(1) As String
Dim propVal(1) As String
Dim rBean As TcR.ResultBean

‘Set Variables
propName(0) = "Name"
propVal(0) = "MyReq1"

Set rBean = tcrConnection.createObjectCOM("5.0.3131327",
"RequirementDB", "LAST_MEMBER", propName, propVal)

If (rBean.isSuccess) Then
    MsgBox "Requirement created succesfully"
End If
```

getObjectCOM

DESCRIPTION

Gets the properties of a given object. It is a wrapper around the **getObject** method.

SYNTAX

```
Public Function getObjectCOM ( _
   objectId As String, _
   ByRef propertyNames As String(), _
   ByRef propsInFile As Boolean(), _
   reqChange As Boolean _
) As ResultBean
```

ARGUMENTS

- `objectId (String)`: Object ID of the object.
- `propertyNames (String ())`: String array for the property names.
- `propsInFile (Boolean ())`: Boolean array indicating property names are to be set in a file.
- `reqChange (Boolean)`: Boolean to indicate locking the object.

RETURNS

A **ResultBean** object from Systems Architect/Requirements Management.

EXAMPLE

```
‘Define variables
Dim propName(1) As String
Dim propInFile(2) As Boolean
Dim obj As Object
Dim rBean As TcR.ResultBean

‘Set variables
propName(0) = "Name"
propInFile(0) = False
propInFile(1) = False

Set rBean = tcrConnection.getObjectCOM("1.0.183339",
propName, propInFile, False)

Set obj = rBean.getResult()
MsgBox obj.getValue("Name")
```

setObjectsCOM

DESCRIPTION

Sets the object properties in Systems Architect/Requirements Management. It is a wrapper around the **setObjects** method.

SYNTAX

```
Public Function setObjectsCOM ( _
   ByRef databeans As DataBean() _
) As ResultBean
```

ARGUMENTS

- `databeans (DataBean ())`: Array of **DataBean** objects.

RETURNS

A **ResultBean** object from Systems Architect/Requirements Management.

EXAMPLE

```
‘Define variables
Dim dBArray(1) As DataBean
Dim dBean As New TcR.DataBean
Dim rBean As TcR.ResultBean

‘Set variables
dBean.objectId = "1.0.151747"
dBean.setValue "Name", "MyReq123"

Set dBArray(0) = dBean

Set rBean = tcrConnection.setObjectsCOM(dBArray)
```

createLinksCOM

DESCRIPTION

Creates trace links with the given type in Systems Architect/Requirements Management. It is a wrapper around the **createLinks** method.

SYNTAX

```
Public Function createLinksCOM ( _
   from As String, _
   ByRef to As String(), _
   linkType As String, _
   subType As String _
) As ResultBean
```

ARGUMENTS

- `from (String)`: Object ID to indicate the beginning of links.
- `to (String ())`: String array for object IDs to indicate the end of links.
- `linkType (String)`: String for the trace link type.
- `subType (String)`: Subtype of link to create. If null, the base type is used.

RETURNS

A **ResultBean** object from Systems Architect/Requirements Management.

EXAMPLE

```
‘Define variables
Dim toLink(0) As String
Dim rBean As TcR.ResultBean

‘Set Variables
toLink(0) = "1.0.151747"

Set rBean = tcrConnection.createLinksCOM("1.0.151761", toLink, "Defining", "")
```

getResultCOM

DESCRIPTION

Gets the output from the result bean. It is a wrapper around the **getResult** method.

SYNTAX

```
Public Function getResultsCOM ( _
output As Object, _
databeans to As DataBean() _
)
```

ARGUMENTS

- `output (Object)`: Result if the return type is String or DataBean, else null.
- `databeans (DataBean())`: Array of DataBean collection if the return type is DataBean collection, else null.

RETURNS

None.

EXAMPLE

```
‘Define variables
Dim toLink(1) As String
Dim rBean As TcR.ResultBean
Dim output As Object
Dim beans() As DataBean

‘Set Variables
toLink(0) = "1.0.151747"
Set rBean = tcrConnection.createLinksCOM("1.0.151761", toLink, "Defining", "")
rBean.getResultCOM output, beans
```

In this example, the **createLinksCOM** method returns a result bean that contains a collection of DataBean, which is now available in the beans array.

getPropertiesCOM

DESCRIPTION

Gets the property names and values from the DataBean.

SYNTAX

```
Public Function getPropertiesCOM ( _
name  As String(),  _
value to As String() _
)
```

ARGUMENTS

- `name (String())`: String array for the property names present in the DataBean.
- `value (String())`: String array for the property values present in the DataBean.

RETURNS

None.

EXAMPLE

```
‘Define variables
Dim toLink(1) As String
Dim rBean As TcR.ResultBean
Dim name  As String()
Dim value() As String()
Dim dBean As DataBean

‘Set Variables
toLink(0) = "1.0.151747"
Set rBean = tcrConnection.createLinksCOM("1.0.151761", toLink, "Defining", "")
rBean.getResultCOM output, beans
Set dBean = beans(0)
dBean.getPropertiesCOM name, value
```

In this example, the **getPropertiesCOM** method returns two arrays, one array containing property names and the other array containing the values for the property names.

NOTES

Similarly, you can use the **getChangeFlagsCOM** method on DataBean to get the change flags.

getDataBeansCOM

DESCRIPTION

Gets the data beans from the DataBean dictionary.

SYNTAX

```
Public Function getDataBeansCOM ( _
beans  As DataBean() _
)
```

ARGUMENTS

- `beans (DataBean())`: Data beans from the DataBean dictionary.

RETURNS

None.

EXAMPLE

```
‘Define variables
Dim toLink(1) As String
Dim rBean As TcR.ResultBean
Dim rd As DataBeanDictionary
Dim beans() As DataBean

‘Set Variables
toLink(0) = "1.0.151747"
Set rBean = tcrConnection.createLinksCOM("1.0.151761", toLink, "Defining", "")
Set rd = rBean.getResultDictionary()
rd. getDataBeansCOM beans
```

In this example, the **getDataBeansCOM** method returns an array (beans) with all the data beans from the DataBean dictionary.

runActivatorCOM

DESCRIPTION

Run an activator's Tcl script.

SYNTAX

```
Public Function runActivatorCOM ( _
activatorID As String, _
ByRef selectedIds As String() _
) As ResultBean
```

ARGUMENTS

- `activatorID (String)`: Object ID of the activator.
- `selectedIds (String ())`: String array of Systems Architect/Requirements Management object IDs or other values. This array is available in Tcl as a global list named "selected".

RETURNS

A ResultBean object from Systems Architect/Requirements Management. The result value is the string specified in the Tcl return statement.

EXAMPLE

```
'Define variables
Dim selectedIds(1) As String
Dim rBean As TcR.ResultBean
Dim tclReturnValue As Object

'Set Variables
selectedIds (0) = "Hello"
'Run the activator named Hello World in MyProject
Set rBean = tcrConnection.runActivatorCOM("\\MyProject\Activators\HelloWorld",
              selectedIds)
Set tclReturnValue = rBean.getResult()
```

Chapter 5: Using the Tcl Scripting API to Access the API

This chapter contains information on using the Tcl API client library to access the API. If you are developing your client application in Java, you can use the Java client library included in the toolkit.

The list of Java API functions is provided in the API Javadoc. The API Javadoc describes each function along with the response expected from the server. Additionally, the list of Property Names, Lists, Keywords and other Constants are defined in the DataBean class in API Javadoc.

You can access the Javadoc from Systems Architect/Requirements Management home page.

1. Click **API Javadoc**.
2. Click the **com.edsplm.tc.req.databeans** package link.
3. From the Class Summary list, click the **DataBean** link.

The constants are defined in a HTML table.

The list of Java API functions is provided in the API Javadoc. The API Javadoc describes each function along with the response expected from the server. Additionally, the list of Property Names, Lists, Keywords and other Constants are defined in the DataBean class in API Javadoc.

You can access the Javadoc from Systems Architect/Requirements Management home page.

1. Click **API Javadoc**.
2. Click the **com.edsplm.tc.req.databeans** package link.
3. From the Class Summary list, click the **DataBean** link.

The constants are defined in a HTML table.

Introduction

Tcl (*Tool Command Language*) is an interpreted language, originally developed by Professor John Ousterhout at the University of California, Berkeley. The Tcl interpreter is freely available on the Internet. Tcl has been ported to most popular operating systems (Microsoft Windows, many UNIX variants, and Macintosh) and hardware platforms.

Tcl is similar to other scripting languages like the Bourne shell, C shell, and Perl. These scripting environments let you execute other programs, providing enough programmability to integrate existing tools into a new tailored application that fits your needs. As an internal macro-like language, Tcl lets you create small applications that automate routine sets of commands.

Systems Architect/Requirements Management includes JACL, a Tcl interpreter written in Java. JACL supports the basic Tcl language (Tcl version 8.0), but does not support Tk, the graphical user interface companion to Tcl.

There are many Internet resources for additional Tcl/Tk information. This is a good starting place:

http://www.tcl.tk

There are number of Tcl books available in bookstores that carry a good selection of computer related technical books. A partial list follows:

- *Practical Programming in Tcl and Tk*, Brent Welch; Prentice Hall
- *Tcl and the Tk Toolkit*, John K. Ousterhout,; Addison-Wesley
- *Tcl/Tk in a Nutshell*, Paul Raines & Jeff Tranter, O'Reilly & Associates
- *Tcl/Tk Pocket Reference*, Paul Raines, O'Reilly & Associates

A wide variety of Tcl extensions are freely available in the public domain. However, these have not been tested with Systems Architect/Requirements Management and are not supported in any way by Siemens PLM Software.

Executing Tcl Scripts

Tcl scripts can be executed in two ways:

- Tcl scripts stored in activator objects can be triggered when a specified event occurs. (See chapter Using Activators.)
- Java API clients can execute Tcl scripts using the **RequirementService** methods **runActivator** and **runScript**.

Transaction Management

Tcl scripts run in a single database transaction. If an error occurs, all changes made by the script are rolled back. When activators are triggered by a **RequirementService** call, the activators run in the same transaction as the API method. So if an error occurs in any activator, the changes made by all the activators and the API call are rolled back.

Parameter Types

In addition to the standard Tcl parameter types, Systems Architect/Requirements Management Tcl has these addition types:

- **OBJECT:** TcR database object identifier.
- **OBJECT_LIST:** A Tcl list of database object identifiers.

A Systems Architect/Requirements Management database object can be identified by its unique database ID (LOID). Objects in the administration module may also be identified by name. The following naming convention is used:

*projectName**administrationFolderName**objectName*

For example, the Requirement type definition in Project1 can be identified with:

```
\\Project1\Type Definitions\Requirement
```

The *administrationFolderName* in this syntax refers to the special predefined folders that appear just beneath a project in the administration module. Subfolders below that level are not recognized in this syntax. For example, if within the **Activators** folder there is a **Macros** folder, and under that a **Get Input** macro, its name would be *projectName***Activators****Get Input**, without the Macros level.

Listing of Tcl Methods

The sections that follow detail Tcl methods available to the Tcl Scripting API.

calculateProperties

DESCRIPTION

Calculates the numeric properties in objects. The values are calculated from all member objects. If it is a tree, it is calculated bottom up, using a formula at every parent object, recursively.

SYNTAX

```
calculateProperties objects properties
```

ARGUMENTS

- `object`: OBJECT_LIST – objects selected for calculation.
- `properties`: OBJECT_LIST – numeric properties, which has formula.

RETURNS

Void, the calculated values are directly set on the objects.

EXAMPLES

```
calculateProperties $objectORobjectList $properties
```

changeApproval

DESCRIPTION

This method is used to update the status of a change approval object. This method is used to submit a requirement or building block for approval or update the status of an existing change approval object. It can be used to force a change of change approval status of an object already being routed. This method is independent of the four out-of-the-box activators shipped with Systems Architect/Requirements Management, Change Approved, Change Rejection, Change Response, and Change Submit.

SYNTAX

```
changeApproval objectIds action comment subject
```

ARGUMENTS

- `objectIds`: OBJECT_LIST– The object ids (LOID) of the change approval objects. The object ids (LOID) of the change approval objects, or (when the action is Change Submitted) the LOIDs of the objects being submitted for approval.
- `action`: STRING_LIST – List of string identifying actions for the objectIds (**Change Rejected**, **Change Approved**, and **Change Submitted** are the only valid actions. Use only one of these actions). This list has a one to one correspondence with objectIds.
- `comment`: STRING_LIST – List of string identifying user comments for the objectIds. The comment applies to this change approval object by this user. This list has a one to one correspondence with objectIds.
- `subject`: STRING – The subject line of the email.

RETURNS

The updated objects.

NOTES

The lists (objectIds, action, and comment) must have the same number of entries. The **changeApproval** call fails if multiple objectIds and one action and comment are passed.

EXAMPLES

```
#get loid of the Change Approval Object.
set objectId [list $currentObject]

set action [list "Change Rejected"]

set comment [list "Comment for $objectId"]

changeApproval $objectId $action $comment $subject
```

copyObjects

DESCRIPTION

Copies the source objects to the specified destination. In addition to the source objects, all their descendents, owned objects and links are also copied.

SYNTAX

```
copyObjects sources destination deep
```

ARGUMENTS

- `sources`: OBJECT_LIST – objects to copy.
- `destination`: OBJECT – owner of copies.
- `deep`:BOOLEAN – If true, copies all descendants of the sources. If false, copies the source objects only.

RETURNS

OBJECT_LIST—The copied objects.

NOTES

Some operations are performed using copy and paste in the Architect/Requirements client but are not possible using the copyObjects API.

For example, you can copy and paste an object into a group in the Architect/Requirements client, but you must use the createLinks method when using the API.

EXAMPLES

```
set copies [copyObjects $requirements $folder]
```

createAction

DESCRIPTION

Allows macros or activators executed on Systems Architect/Requirements Management servers to initiate Systems Architect/Requirements Management client actions, for example, launch workstation applications.

`createAction FileDownload` allows macros or activators executed on Systems Architect/Requirements Management servers to download a file to the Systems Architect/Requirements Management client workstation, and optionally open it in an application. See createAction FileDownload.

SYNTAX

```
createAction actionClass [parameter or list_of_parameters]

createAction LaunchWebBrowser urlString
createAction LaunchWebBrowser [list urlString sessionVarName sessionID]
createAction RunMacro [list projId macroName selected]
createAction RunReport [list testReport startObjectID]
createAction RunReport [list testReport]
createAction GoToAndOpen objectId
createAction GoToAndOpen [list objectId OpenCommand]
createAction GoToAndOpen [list objectId OpenCommand readOnly]
createAction GoToAndOpen [list objectId OpenCommand readOnly moduleName]
createAction Refresh
```

ARGUMENTS

- `actionClass`: STRING – Identifies the client action to be taken.

  `LaunchWebBrowser`: Open the given URL in a browser.

  `RunMacro`: Run the named macro from the given.

  `RunReport`: Run saved reports from Macros or Activators.

  `GoToAndOpen`: Navigate to an object and optionally open it.

  `Refresh`: Refreshes the client as though the user pressed the **Refresh** button on the client toolbar.

- `urlString`: STRING – URL to open in the browser.
- `sessionVarName:=`: STRING – optional mechanism to allow the Systems Architect/Requirements Managements client to pass a sessionID to a JSP, avoiding the need for the JSP to perform its own Teamcenter Requirements login. This is the name of an argument to pass to the JSP. The Systems Architect/Requirements Managements client appends an argument by that name to the URL before sending it to the web browser.
- `sessionID`: STRING – optional mechanism for the JSP to obtain the argument to be used for the sessionID `sessionVarName` above. The value obtained (from the Systems Architect/Requirements Management server) for `sessionID` is assigned to `sessionVarName`.
- `projId`: STRING – objectId (LOID) for the project in which the macro is found.
- `macroName`: STRING – The name of a Macro to run in that project.
- `selected`: STRING – This value is set as the `selected` global Tcl variable when the macro runs. This argument replaces the normal value of selected, which is a list of the selected objects in the

user interface. This argument is optional. If it is not provided, the normal value for `selected`, the selected objects, is used.

- `testReport`: STRING – The name of the report.
- `startObjectID`: STRING – start object for search.

  If there is no `startObjectID` specified, the action class uses the currently selected object or saved location as start object. The action class displays all reports in the project and lets the user pick one if the specified report does not exist in the project.
- `objectId`: STRING – The ID for the object to become selected.
- `OpenCommand`: STRING – Keyword that selects whether to simply go to the object, or to go to the object and open it.

  To go to the object and open it, use `GoToAndOpen`. The object is opened in the application that is used when the `Open` command is used on the object.

  To go to the object without opening it, use `GoTo`.

  If `OpenCommand` is not specified, the default is `GoTo`.
- `readOnly`: STRING – `true`if the object to be opened should be opened in read-only mode. Otherwise, `false`.

  If `readOnly` is not specified, the default is `false`.
- `moduleName`: STRING – Identifies whether to go to the Administration Module or the User Module.

  Use `tcrAdmin` for the Administration module.

  Use `tcr` for the user module.

  If `moduleName` is not specified, the default is `tcr`.

RETURNS

None

NOTES

Calling **createAction** adds information to the transaction result that requests certain actions to occur at the Systems Architect/Requirements Management client. These actions occur only when the client initiates the request. If the action came from Systems Architect/Requirements Management's Excel Live or Visio Live, or from a JSP calling the Requirements Service API, the requested action does not occur. For example, a **Create** activator might call **createAction RunMacro** to prompt the user for additional information on the newly created object. If the new object is created through Excel or Visio, the macro will not be run.

EXAMPLES

There are two ways to call the **LaunchWebBrowser** action class.

- Launch Internet Explorer and navigate to a specified URL:

  ```
  set url http://www.ugs.com
  createAction LaunchWebBrowser $url
  ```
- Launch Internet Explorer and pass in the user name and session to a Systems Architect/Requirements Management custom JSP in order to avoid login when accessing Systems Architect/Requirements Management objects:

```
set url “http://tcr_server_url:8080/tcr/..../xxx.jsp”
set jspParmName tcrSessionId
createAction LaunchWebBrowser [list $url $jspParmName]
```

The `userplusid` is the parameter name for user id and session. The jsp should use the parameter value. To avoid this from happening login as follows:

```
String tcrID = request.getParameter("tcrSessionId");
ResultBean result = RequirementService.getList(tcrID, null,
DataBean.LIST.PROJECT, new String[] {DataBean.ALL_PROPERTY}, 1, 0);
```

- The following is an example for the **RunMacro** action class:

  `createAction RunMacro [list $currentProject` *`macroName`*`]`

  where *macroName* is an existing macro name in the current project.

- The following is an example for the **GoToAndOpen** action class:

  `createAction GoToAndOpen [list $myObject GoToAndOpen]`

  This example navigates to the object `myObject`, and opens the object for edit.

- The following is an example for the **GoToAndOpen** action class:

  `createAction GoToAndOpen [list $myActivator GoToAndOpen tcrAdmin]`

  This example navigates to the object `myActivator` in the Administration module and opens the object for edit.

The following is an example of how to obtain the `sessionID` from the server in your macro or activator:

```
set sessionID [setEnvironment SessionID]
createAction LaunchWebBrowser [list urlString sessionVarName $sessionID]
```

createAction FileDownload

DESCRIPTION

Allows macros or activators executed on Architect/Requirements servers to download a file to the Architect/Requirements client, and optionally open it in an application. Also allows optionally opening a local client file without downloading from the server.

SYNTAX

```
createAction FileDownload [list serverFile app localFilePath isLive]
```

ARGUMENTS

- `serverFile`: STRING — (required) The following options are available:
    - The path of the file on server that the user wants to download.
    - The keyword OpenLocalFileOnly. When that keyword is used, the file is not downloaded from the server, but already exists on the client machine.
- `app`: STRING — The following options are available:
    - If this parameter is not present, the file is downloaded and a message is shown providing the location. No application is launched.
    - If the value of this parameter is keyword, **Default**, the downloaded file is opened in Internet Explorer.
    - If the value of this parameter is keyword, **MS_EXCEL** or **EXCEL**, the file is opened in Microsoft Excel.
    - If the value of this parameter is keyword, **MS_WORD**, the file is opened in Microsoft Word.
    - If the value of this parameter is keyword, **VISIO**, the file is opened in Microsoft Visio.
    - In all other cases, the file is opened using the user-specified application. The complete path name to the executable file must be specified.
    - If the value of this parameter is keyword, **NO**, then the file is not opened, and the user is not notified that file download has successfully completed.
- `localFilePath`: STRING — The local file path. If this parameter is missing or contains a `""` string, a temporary file is created.

  If this parameter is a simple file name, then that file name is used when writing to a temporary path. The temporary path is the same path that is retrieved when using the **ClientJavaAPI.getTempDir** function, which can be called from external Java code (run via **createAction** RunJava).
- `isLive`: STRING — This is `true` if the file is to be opened live. Otherwise the value is `false`. If this parameter is missing or contains a `""` string, the value of `false` is used.

RETURNS

None

NOTES

If the **localFilePath** argument is omitted and the default **TcSE-nnnn** temp file name is used, the client file gets deleted when the user logs out.

EXAMPLES

- When the activator runs, the file **D:/XYZ.ppt** is downloaded from the server and is opened using Internet Explorer:

```
set serverFile "D:/XYZ.ppt"
set application Default
createAction FileDownload [list $serverFile $application]
```

- When the activator runs, the file **D:/XYZ.ppt** is downloaded from the server and is opened using the user-selected application, PowerPoint:

```
set serverFile " D:/XYZ.ppt "
set application {C:\win32app\Microsoft
   Office\Office10\POWERPNT.EXE}
createAction FileDownload [list $serverFile $application]
```

- When the activator runs, the file **D:/XYZ.ppt** is downloaded from the server and is opened using the user-selected application, Excel:

```
set serverFile " D:/XYZ.ppt "
set application “EXCEL”
createAction FileDownload [list $serverFile $application]
```

- When the activator runs, the file **D:/XYZ.vsd** is downloaded from server, copied to **D:/ABC.vsd**, and it is opened using the user-selected application, Visio:

```
set serverFile "D:/XYZ.vsd"
set clientFile “D:/ABC.vsd”
set application {C:\win32app\Microsoft Office\Visio11\VISIO.EXE}
createAction FileDownload [list $serverFile $application $clientFile]
```

- Or the application keyword can be used: When the activator runs, the file D:/XYZ.vsd is downloaded from server, copied to D:/ABC.vsd, and it is opened using the user-selected application, Visio:

```
set serverFile "D:/XYZ.vsd"
set clientFile “D:/ABC.vsd”
set application “VISIO”
createAction FileDownload [list $serverFile $application $clientFile]
```

- When the activator runs, no file is downloaded from server, local file D:/ABC.mht is opened Live using the user-selected application, Excel:

```
set serverFile " OpenLocalFileOnly"
set clientFile “D:/ABC.mht”
set application “EXCEL”
set isLive “true”
createAction FileDownload [list $serverFile $application $clientFile
   $isLive]
```

createAction RunJava

DESCRIPTION

Allows identification of Java classes and execution of methods from those classes in activators, macros, or custom menu items.

- **createAction RunJava** executes your Java code in the Architect/Requirements client process. Faulty code can cause the client to freeze, abort, consume excessive memory, or become unstable. The burden is on the Java developer to avoid such consequences.
- Although **createAction RunJava** runs Java code in the Architect/Requirements client, this is not intended as a customization point to add or modify client behavior, and there is no client Java API. This is intended as an interface point to aid in integrating Architect/Requirements with other client applications, and is supported only in that context.

SYNTAX

```
createAction RunJava [list className methodName]
createAction RunJava [list className methodName requiresOfficeLive]
createAction RunJava [list className methodName requiresOfficeLive parameter1
parameter2 ...]
```

ARGUMENTS

- `className`: STRING — The name of the class containing the method to run.
- `methodName`: STRING — The name of the method in the class to be run.
- `requiresOfficeLive`: STRING — `true` if Office Live installation is required to run the method. Otherwise, `false`.

  The Java file may be a wrapper around C# code that is communicating with the client through the socket, and thus may require that the Office Live interface be installed to function properly. If this parameter is not used, the value is `false`.
- *`parameter1, parameter2, etc.`*: STRING — Values to be passed to the Java method being called.

RETURNS

None

NOTES

- To run methods from the **.jar** file, the class must always include a method with this signature:

  ```
  public String connectTcSE(String controllerPath, String serverIP,
  String socketServicePort, String sessionID)
  ```

  This method provides the necessary parameters needed for the external application to connect back with Architect/Requirements. To connect back to the client, the first three parameters can be used. To connect back to the server, the last can be used.
- Only public methods can be called.
- Valid return types for the Java methods are `void`, `String`, and `String[]`. Any values returned from the method are printed to the log file.

- The method parameters must either be empty (no parameters) or must be a single parameter of type `String[]`.
- The location of the **.jar** file must be identified to Architect/Requirements via the **Package.Location** parameter set by the administrator.

EXAMPLES

For source code related to these examples, see Using createAction RunJava in chapter **Unsatisfied xref reference**, ***Unsatisfied xref title***.

- When the activator runs, it gets a list of all the public methods in the class:

  ```
  createAction RunJava [list TestRunJavaClass ShowMethods]
  ```

- When the activator runs, it runs the method named **doNothing**, which prints a message to the log and displays a dialog for the user to click:

  ```
  createAction RunJava [list TestRunJavaClass doNothing]
  ```

- When the activator runs, it verifies that the live Office interface is installed before running the **doNothing** method:

  ```
  createAction RunJava [list TestRunJavaClass doNothing true]
  ```

- When the activator runs, it displays a **Hello World!** dialog:

  ```
  createAction RunJava [list TestRunJavaClass printHelloWorld false arg1]
  ```

- When the activator runs with the class name misspelled, it returns a message indicating that the class could not be found:

  ```
  createAction RunJava [list TestRunJavaClas doNothing]
  ```

- When the activator runs with the method name misspelled, it logs and returns a message indicating that the method could not be found:

  ```
  createAction RunJava [list TestRunJavaClass doNothin]
  ```

- When each of these activators runs, there are no errors:

  ```
  createAction RunJava [list TestRunJavaClass printHelloWorld false arg1]
  createAction RunJava [list TestRunJavaClass doNothing false arg1 arg2 arg3]
  createAction RunJava [list TestRunJavaClass doNothingString]
  createAction RunJava [list TestRunJavaClass doReturnStringArray]
  ```

- When each of these activators runs, it results in errors and a return message indicating that the method could not be found:

  ```
  createAction RunJava [list TestRunJavaClass returnInt]
  createAction RunJava [list TestRunJavaClass doSomething false 21]
  createAction RunJava [list TestRunJavaClass doSomething false arg1 arg2 arg3]
  createAction RunJava [list TestRunJavaClass printIt whatever]
  ```

- When Java runs on the client that calls a server macro:

  ```
  createAction RunJava [list TestRunJavaClass runTestMacro false
  [getValue $selected Name]]
  ```

  You must create a macro named **HelloWorldMacro** for the command to work. Add the following code to the macro:

  ```
  displayMessage "Hello World, From Macro, selected: $selected"
  return "Hello World (Result)"
  ```

The example demonstrates passing information to and from the client. The name of the selected object is passed to a Java method running on the client and back to a macro running on the server where the name is displayed. The return value from the macro is passed to the client. The client then displays the return value.

createBaseline

DESCRIPTION

Creates a baseline containing the specified objects.

SYNTAX

```
createBaseline objects name
```

ARGUMENTS

- `objects`: OBJECT_LIST – List of versionable objects to baseline.

  `name`: STRING – The new baselines name.

RETURNS

Void

NOTES

Versionable object types are Requirements and Building Blocks.

EXAMPLES

```
createBaseline [getList $folder MEMBER_LIST] “Version 1”
```

createExternalLink

DESCRIPTION

Creates a Teamcenter Interface (WOLF) link from Systems Architect/Requirements Management to an object in an external application such as Teamcenter Engineering or Teamcenter Enterprise. This API creates a Systems Architect/Requirements Management database entry to record the link, but its operation is entirely internal to Systems Architect/Requirements Management.

createExternalLink does not create such a link in the external application. It does not make any call to notify the external application of the existence of the link. To create such a link in the external application, you must invoke an API of that application, either before or after calling **createExternalLink**.

createExternalLink can be used only for creating original WOLF links; it cannot be used to create new proxy links.

SYNTAX

```
createExternalLink GUID OID name icon biDirectional tcrObject description
```

ARGUMENTS

- `GUID`: STRING – Teamcenter Interface external application identifier.
- `OID`: STRING – Teamcenter Interface external object identifier.
- `name`: STRING – Name of the external object.
- `icon`: OBJECT – This argument is reserved for future use.
- `biDirectional`: BOOLEAN – This argument is reserved for future use.
- `tcrObject`:OBJECT – Systems Architect/Requirements Management end of the WOLF link.
- `description`: STRING – Description of the link.

RETURNS

OBJECT–The new WOLF handle object.

createLinks

DESCRIPTION

Creates trace links, generic links, or connections from one object to a list of objects.

SYNTAX

```
createLinks from to [linkType] subtype
```

ARGUMENTS

- `from`: STRING – start of links or connections.
- `to`: OBJECT_LIST – end of links or connections.
- `linkType`: STRING – type of tace link, generic link, or connection to create, default Complying, see examples and DataBean.LINK.
- `subtype`: STRING – subtype of trace link, generic link, or connection to create, if null the base type is used.

RETURNS

OBJECT_LIST–The new trace links, generic links, or connections.

EXAMPLES

- To create trace link where *from* is defining object and *to* is complying:

```
createLinks $from $to
set newLinks [createLinks $defining $complyingObjs Complying $subType]
```

- To create trace link where *from* is complying object and *to* is defining:

```
createLinks $from $to Defining
```

- To create connection beginning at *from* and ending at *to*:

```
createLinks $from $to Connection
```

- To create connection in the reverse direction:

```
createLinks $from $to {Defining Connection}
```

- To create generic link beginning at *from* and ending at *to*:

```
createLinks $from $to {Outgoing Generic Link}
```

- To create generic link in the reverse direction:

```
createLinks $from $to {Incoming Generic Link}
```

SEE ALSO

`createObject`

createObject

DESCRIPTION

Creates a design object in the database. This command does not create a trace link or project, those are separate commands.

SYNTAX

```
createObject name owner type [position]
```

ARGUMENTS

- `name`: STRING – Name of the new object, if blank, a unique name is generated.
- `owner`: OBJECT – Owner, or sibling, of the new object.
- `type`: STRING – Type, or subtype, of the new object
- `position`: STRING – position of the new object relative to owner. See DataBean.POSITION.

RETURNS

OBJECT– The new Object. A Tcl error if objects of the give type can not be owned by subordinate to objects of the given owners type.

NOTES

Valid positions are LAST_MEMBER, FIRST_MEMBER, LAST_SIBLING and NEXT_SIBLING. Systems Architect/Requirements Management schema objects may also be created. These objects are normally only created by a Systems Architect/Requirements Management project administrator.

EXAMPLES

```
# Create a requirement as a sibling of a given requirement
createObject "" $Req Requirement NEXT_SIBLING
# Create a Note named "mynote" with the "Rationale" subtype
set newNote [createObject "mynote" $req Rationale]
```

createProject

DESCRIPTION

Creates and initialize a new Systems Architect/Requirements Management project.

SYNTAX

```
createProject [name]
```

ARGUMENTS

- `name`: STRING – the new projects name, if not provided an unique default name is generated.

RETURNS

OBJECT–The new project object.

NOTES

Project administration privilege level is required to use this command.

EXAMPLES

```
set proj [createProject {My Project}]
```

createShortcuts

DESCRIPTION

Creates a new Shortcut to the existing object.

SYNTAX

```
createShortcuts orig owner position
```

ARGUMENTS

- `orig`: OBJECT_LIST – The original objects for which the shortcuts are to be created.
- `owner`: OBJECT – The owner (or sibling) of the shortcut.
- `position`: STRING – Position of new shortcut relative to owner, see DataBean.POSTITION. This argument is optional. It defaults to LAST_MEMBER.

RETURNS

- `scObjects`: The new shortcut objects.

EXAMPLES

```
createShortcuts $requirement $folder LAST_MEMBER
```

createUser

DESCRIPTION

Creates a new Systems Architect/Requirements Management user object.

SYNTAX

```
createUser owner name password maxPrivilege
```

ARGUMENTS

- `owner`: OBJECT – Project that the user belongs to.
- `name`: STRING – The new projects name, if not provided a unique default name is generated.
- `Password`: STRING – The user's initial password.
- `maxPrivilege`: STRING – The maximum privilege the user can have to any project. See DataBean.USER_PRIVILEGE.

RETURNS

OBJECT–The new user object.

NOTES

If the owner is null the Administration Project is used. All user objects are created in the Administration project. If the owner is another project, the user is added to that project. If a user with the given name already exists then a new user object is not created, but the user is added to the given project.

EXAMPLES

```
set user [createUser $project "fred" "" "Read and Write"]
```

createVariant

DESCRIPTION

Creates a variant of a versionable object.

SYNTAX

```
createVariant object
```

ARGUMENTS

- `object`: OBJECT – The object.

RETURNS

OBJECT–The new variant.

EXAMPLES

```
set variant [createVariant $Req]
```

createVersion

DESCRIPTION

Creates a version of a versionable object.

SYNTAX

```
createVersion object
```

ARGUMENTS

- `object`: OBJECT – The object to version.

RETURNS

OBJECT—The new version.

EXAMPLES

```
set version [createVersion $Req]
```

deleteLinks

DESCRIPTION

Remove links between one object and a list of objects.

SYNTAX

```
deleteLinks from to [linkType] subtype
```

ARGUMENTS

- `from`: OBJECT – Start of links.
- `to`: OBJECT_LIST – End of links.
- `linkType`: STRING – Type of link to delete, default Complying, see DataBean.LINK.
- `subtype`: OBJECT - Type definition ID for the subtype of link to delete. If this is null, any subtype may be deleted. This argument is optional.

RETURNS

OBJECT_LIST–The deleted links.

NOTES

`linkType` may be Defining, Complying, Connection, Uses, Group, Incoming Generic Link, or Outgoing Generic Link.

A link can also be deleted using a `deleteObjects` call on the link itself. In some cases, there can be multiple links of the same type between two objects. This causes `deleteLinks` to fail due to ambiguity. `deleteObjects` should be used in these cases.

EXAMPLES

```
deleteLinks $from $to
set newLinks [deleteLinks $defining $complyingObjs Complying $subType]
```

SEE ALSO

`restoreFromTrashcan`

deleteObjects

DESCRIPTION

Deletes Systems Architect/Requirements Management objects and moves them to trashcan. Objects that are already deleted are destroyed.

SYNTAX

```
deleteObjects objects
```

ARGUMENTS

- `objects`: OBJECT_LIST – The objects to delete or destroy.

RETURNS

OBJECTS_LIST–Successfully deleted objects or VOID if destroying objects.

NOTES

Be very careful using **deleteObjects**. For most objects, deletions occur without any further confirmation. The extent of the **deleteObjects** action is the same as if it were done from the user interface. The action for **deleteObjects** is subject to the Access Control settings in the database. When you delete an object (either directly, or because it was a descendent of a deleted object or owned by a deleted object), subsequent attempts to reference it may generate a Tcl error.

When an object is deleted using the Systems Architect/Requirements Management client, a confirmation message window is displayed. For some objects, a second confirmation message window is displayed. The second message warns about special conditions, such as not being able to undo the delete. Tcl developers should confirm in these cases, using **setEnvironment SetResponse**. Projects and other schema objects require **OPT_PARAM_DELETE_PROJECT** set to **Yes** in order to work.

For example:

```
# delete the project
setEnvironment SetResponse Yes OPT_PARAM_DELETE_PROJECT
deleteObjects $project
```

See the **setEnvironment** command for additional details.

Using **deleteObjects** on an object that is already deleted causes it to be removed from the recycle bin. This means, calling **deleteObjects** twice on the same object causes it to be destroyed rather than left in the recycle bin.

EXAMPLES

```
deleteObjects $objList
```

displayMessage

DESCRIPTION

Displays a message in the Systems Architect/Requirements Management client.

SYNTAX

```
displayMessage message [type] [object]
```

ARGUMENTS

- `message`: STRING – The message to display.
- `type`: INTEGER – See MessageBean.
    - 4 – INFO (default) display message in popup information box.
    - 5 – WARNING Display message in popup warning box.
    - 6 – ERROR Display message in popup error box.
- `object`: OBJECT – Database object to attach to the message. The object name is added to the message.

RETURNS

VOID.

NOTES

Messages are displayed after the current transaction has completed. Popup message windows remain until you click **OK**.

EXAMPLES

```
displayMessage "Hello World"
displayMessage "Message" 4 $req
```

emptyTrashcan

DESCRIPTION

Destroys the objects in the current user's trashcan.

SYNTAX

```
emptyTrashcan
```

ARGUMENTS

- None

RETURNS

VOID

EXAMPLES

```
emptyTrashcan
```

export2Excel

DESCRIPTION

Exports a Systems Architect/Requirements Management object to a Microsoft Excel file.

SYNTAX

```
export2Excel objects properties isAlive templateID level
   relationship [sessionID]
```

ARGUMENTS

- `objects`: OBJECT_LIST – List Systems Architect/Requirements Management objects to export.
- `properties`: STRING_LIST – The properties to export for each object. This value is unused if a `templateID` argument is provided.
- `isAlive`: BOOLEAN – If true the Excel file can be used in an Excel live session.
- `templateID`: OBJECT – The Excel template or saved view used to format the output. This value must be { } or "" if a properties list is provided to format the output.
- `level`: INTEGER_LIST – Indentation level of corresponding object. This value is used only when the `templateID` argument identifies an Excel template object. See Notes for additional details.
- `relationship`: STRING_LIST – Relationship between corresponding object and its superior. This value is used only when the `templateID` argument identifies an Excel template object. See Notes for additional details.
- `sessionID`: STRING – Optional session ID.

RETURNS

STRING – Excel file name.

NOTES

Either a `properties` list or a `templateID` must be provided. If both are provided, then the `templateID` argument is used and `properties` is ignored.

If an Excel template is identified in the `templateID` argument and the level and/or relationship lists are provided, they must each be the same length as the objects list. Entries in level and relationship are matched to the corresponding Systems Architect/Requirements Management object in the objects list. If the Excel template does not have level or relationship rules, then the level and/or relationship arguments can be passed as { } or "".

A unique file name is generated automatically for the excel file. The pathname of the file is returned.

EXAMPLES

Locate a search object and extract the query. The search script is as follows:

```
SELECT Requirement
     FOREACH
         ADD Complying Objects


set search {\\Test\Reports and Formatting\ExcelExampleSearch}
set script [getValue $search Script]
```

Run the search.

```
set result [search $script $selected]
```

Iterate over the first-level search result entries. Entries are lists with 3 elements, the Architect/Requirements object, relationship to the object it is indented under, and a list of sub-entries.

```
foreach entry $result {
     # extract the information from the entry
     set req [lindex $entry 0]
     set relation [lindex $entry 1]
     set subEntries [lindex $entry 2]

     # build lists for the objects, level and relationship arguments for
export2Excel
     lappend objects $req
     lappend relations $relation
     lappend levels 1

     # iterate over the second level entries (complying objects) ,
     # extract the information and add it to the argument lists
     foreach subEntry $subEntries {
          set obj [lindex $subEntry 0]
          set relation [lindex $subEntry 1]
          lappend objects $obj
          lappend relations $relation
          lappend levels 2
     }
}
```

Set the Excel template to use for the export. Its rule table appears as follows:

Level Relationship

{%L-1}

{%L-2} {%R-Complying Objects}

```
set template {Excel Example}
```

Initialize the remaining export2Excel arguments.

```
set live true
set properties {}
```

Export the excel file.

```
set excelFile [export2Excel $objects $properties $live $template $levels
$relations]
displayMessage $excelFile
```

exportDocument

DESCRIPTION

Writes objects from the database to a Word, Excel, AP233 STP (Part 21 Standard), AP233 XML (Part 28 Standard), or Systems Architect/Requirements Management XML file.

SYNTAX

```
exportDocument objectList type outputTemplate deep includeOLE live
```

ARGUMENTS

- `objectList`: OBJECT_LIST – The objects to export.
- `type`:: STRING – The type of file to export. MS_WORD, MS_EXCEL, AP233, AP233_28, XML, SCHEMA, or PROJECT. The default is MS_WORD.
- `outputTemplate`: OBJECT or STRING – The Excel or document template used to format the output. The template can be identified by its object LOID or by its name. The LOID is preferred because it avoids the overhead of searching for a name match.
- `deep`: BOOLEAN – For Word exports, `deep` specifies whether to include the descendents of the selected object(s) in the export. The default is true.
- `includeOLE`: BOOLEAN – For Word exports, `includeOle` specifies whether embedded OLE objects are included in the export. The default is true.
- `live`: BOOLEAN – Applies to Excel exports only. If true, creates an Excel live file. If false, creates a static Excel file. The default is false.

RETURNS

STRING—Path name of the exported file.

NOTES

If a folder is specified in `objectList`, the document template property of the folder is used as the style sheet of the document if the `outputTemplate` argument is omitted.

The exported file types are:

- MS_WORD – Word file in MHTML format.
- MS_EXCEL – Excel file in MHTML format.
- AP233 – AP233 STP file in STEP format, conforming to the Part 21 standard.
- AP233_28 – AP233 XML file, conforming to the Part 28 standard.
- XML – Systems Architect/Requirements Management XML file of the specified objects.
- SCHEMA – XML file of schema objects for the specified project.
- PROJECT – XML file for all of the specified project.
- TC_XML – Export data for migration to Teamcenter.

Exports done through this API are always carried out within the web server process, even if the Systems Architect/Requirements Management's **ExternalImportExport** setting has been configured to use an external process. This could result in performance or memory consumption issues within the web server process.

The file created by `exportDocument` is temporary. It may not have the correct extension and it is deleted automatically after some time. To retain the file, it must be copied or renamed after the export. This can be done using the Tcl `file copy` or `file rename` command.

EXAMPLES

```
set wordFile [exportDocument $folder]
file rename $wordFile {myDocument.mht}
set excelFile [exportDocument $folder MS_EXCEL $excelTemplate]
```

```
create an Excel live file
set excelFile [exportDocument $selected MS_EXCEL "Default Excel Template" false
false true]
```

exportXML

DESCRIPTION

Exports Systems Architect/Requirements Management schema objects or projects to an XML file.

SYNTAX

```
exportXML objects [filename]
```

ARGUMENTS

- `objects`: OBJECT_LIST – List of objects to export.
- `filename`: STRING – Filename for the new XML file. The default is to generate a unique filename.

RETURNS

STRING – the XML file name.

NOTES

If objects contains a single project object then that entire project is exported, otherwise the individual schema objects in the list are exported.

This method is deprecated. Instead, the exportDocument API method is recommended.

EXAMPLES

```
set xmlFile [exportXML $myProject]
```

getEnvironment

DESCRIPTION

Retrieves one of Systems Architect/Requirements Management configuration parameters.

SYNTAX

```
getEnvironment paramName
```

ARGUMENTS

- `paramName`: STRING – Name of the parameter to retrieve.

RETURNS

STRING – the parameter value.

NOTES

The valid parameter names can be seen in the parameter name column of Systems Architect/Requirements Management' Web Application Configuration web page.

EXAMPLES

```
set path [getEnvironment ImportExportDir]
```

getList

DESCRIPTION

Retrieves a list of Systems Architect/Requirements Management objects that are related to the given object by the specified relationship.

SYNTAX

```
getList object listType [depth]
```

ARGUMENTS

- `object`: OBJECT – Object containing or owning the list.
- `listType`: LIST_ENUM – The type of object list, see DataBean.LIST.
- `depth`: INTEGER – Recursive depth to follow relationship. Default is 1.

RETURNS

OBJECT_LIST.

NOTES

The depth argument is only supported when `listType` is MEMBER_LIST, and the given object is a requirement.

The list of Java API functions is provided in the API Javadoc. The API Javadoc describes each function along with the response expected from the server. Additionally, the list of property names, lists, keywords and other constants are defined in the **DataBean** class in API Javadoc.

You can access the Javadoc from the Systems Architect/Requirements Management home page.

1. Click **API Javadoc**.
2. Click the **com.edsplm.tc.req.databeans** package link.
3. From the Class Summary list, click the **DataBean** link.

The constants are defined in a HTML table.

EXAMPLES

```
set members [getList $folder MEMBER_LIST]
```

```
getList $obj {PROPERTY_LIST}
```

Returns the list of actual property objects for the given object. This list includes all user-defined properties. It includes some standard properties, but does not include system-defined properties that are stored as Java data members on the object itself, like Create or Change User and Time.

For each Property instance, these calls will return its name and value:

```
getValue $prop {Name}
```

```
getValue $prop {Current Value}
```

getObject

DESCRIPTION

Returns a list of property values for the given object.

SYNTAX

```
getObject object propertyList
```

ARGUMENTS

- `object`: OBJECT – The Systems Architect/Requirements Management object.
- `propertyList`: STRING_LIST – List of desired properties, see DataBean.PROPERTY.

RETURNS

OBJECT_LIST – Property values for regular calls and for PickList property objects that are stored in the dynamic choice property.

NOTES

In addition to the system properties defined in **DataBean.PROPERTY**, any applicable user-defined properties may be used.

The LOID of a pick list object can be obtained using the `getObject` TcL function. The only needed change is that the property name in these function has a prefix of LOID.

So when you need the values for a property, you simply use the `getObject`. But if your need the LOID(s) of the objects that are stored as values in the PickList choice property, use `getObject` and prefix `LOID:` in front property name. For additional information, see Using A Pick List Activator in chapter **Unsatisfied xref number**, ***Unsatisfied xref title***.

For information on user-defined, dynamic choice property (pick list property, see the section *Setting a Dynamic Choice List for a Choice Property Definition* in the *Systems Architect/Requirements Management Project Administrator's Manual*.

EXAMPLES

```
set values [getObject $Requirement {ROIN Name Text}]
```

If **Assign to** is a dynamic choice property that exists on the object:

- To obtain the value in the **Assign to** property, make a call as follows:

  ```
  getObject $object “Assign To”
  ```

- To obtain the LOID of the object in the **Assign To** property, make a call as follows:

  ```
  getObject $object {"LOID:Assign To"}
  ```

getProjects

DESCRIPTION

Returns a list of Systems Architect/Requirements Management project objects.

SYNTAX

```
getProjects PROJECT_LIST

getProjects ADMIN_VIEW_LIST

getProjects USER_VIEW_LIST
```

ARGUMENTS

- `PROJECT_LIST:` OBJECT_LIST – Get all projects except the admin project.
- `ADMIN_VIEW_LIST:` OBJECT_LIST – Get all projects including admin.
- `USER_VIEW_LIST:` OBJECT_LIST – Get all projects except the admin and include the trash can. This argument is the default if no argument is given.

RETURNS

OBJECT_LIST – Projects to which this user has access.

EXAMPLES

```
set projects [getProjects PROJECT_LIST]

set projects [getProjects ADMIN_VIEW_LIST]

set projects [getProjects USER_VIEW_LIST]
```

getPropertiesWithFormula

DESCRIPTION

Returns a property definition that applies to the given object.

SYNTAX

```
getPropertiesWithFormula objects
```

ARGUMENTS

- `object`: OBJECT – object for which numeric properties with formula is obtained.

RETURNS

OBJECT_LIST–All the Numeric Properties that have formula for the passed objects.

EXAMPLES

```
getPropertiesWithFormula $objects
```

getPropertyDefinition

DESCRIPTION

Returns a property definition that applies to the given object.

SYNTAX

```
getPropertyDefinition object property
```

ARGUMENTS

- `object`: OBJECT – object containing or owning the list.
- `property`: STRING – The property name.

RETURNS

OBJECT– The property definition.

EXAMPLES

```
set propertyDef [getPropertyDefinition $folder Name]
```

getPropertyDefinitions

DESCRIPTION

Returns a list of property definition names that apply to the give object types.

SYNTAX

```
getPropertyDefinitions object types
```

ARGUMENTS

- `object`: OBJECT – type or subtype definition.
- `types`: STRING_LIST – type of property definitions to retrieve. The default value is **All Property**.

RETURNS

STRING_LIST–Property definition names.

NOTES

Hidden properties are those that do not appear in the content pane of the Systems Architect/Requirements Management client. These properties contain information that is not readable.

getRemoteObjectTraceReport

DESCRIPTION

Retrieves the trace report for the remote object. This is used in conjunction with **Proxy linking** between Systems Architect/Requirements Management and Teamcenter Engineering or Teamcenter Enterprise. It is not useful with the original WOLF linking. This action posts a request to the remote system to retrieve the trace report. Hence, there can be a delay at that point.

For troubleshooting, first try using the **Trace Report** UI command. If the **Trace Report** fails then it is likely a configuration problem and not mis-use of the API. Check the Architect/Requirements and Teamcenter Engineering/Teamcenter Enterprise logs for error messages.

SYNTAX

```
getRemoteObjectTraceReport tcrObject
```

tcrObject is the proxy object created when a Teamcenter Engineering/Teamcenter Enterprise object is dragged (or copy/pasted) into Architect/Requirements.

ARGUMENTS

- `tcrObject`: STRING – Proxy object.

RETURNS

- `String traceReport`: Trace report in HTML format.

getValue

DESCRIPTION

Retrieves a value from an object.

SYNTAX

```
getValue object property
```

ARGUMENTS

- `object`: OBJECT – Object containing desired value.
- `property`: STRING – System property name (see DataBean.PROPERTY) or user-defined property name.

RETURNS

The property's value, or an empty string if the property is not found on the object.

NOTES

The LOID of a pick list object can be obtained using the `getValue` TcL function. The only needed change is that the property name in these function has a prefix of LOID.

When you need the values for a property, use the `getValue`. But if your need the LOID(s) of the objects that are stored as values in the Pick list choice property, use `getValue` and prefix `LOID:` in front of the property name.

For more information, see Using A Pick List Activator in chapter **Unsatisfied xref number**, ***Unsatisfied xref title***. For information about setting a dynamic choice list for a choice property, see the *Systems Architect/Requirements Management Project Administrator's Manual*.

The MHTML keyword is used to retrieve the text content of a note or requirement including graphics. For performance reasons the OLE content is not included when the MHTML keyword is used. If the text content is opened in Microsoft Word, or used to set the content of another requirement, the OLE objects will be represented with a graphic, but the OLE application cannot be launched. To include OLE content use the MHTML_FULL keyword instead. The MHTML_FULL keyword is only supported in getValue, always use MHTML in setValue.

For example: # copy the content of a note to a new note including OLE objects

```
setValue $newNote MHTML [getValue $oldNote MHTML_FULL]
```

The list of Java API functions is provided in the API Javadoc. The API Javadoc describes each function along with the response expected from the server. Additionally, the list of Property Names, Lists, Keywords and other Constants are defined in the DataBean class in API Javadoc.

You can access the Javadoc from Systems Architect/Requirements Management home page.

1. Click **API Javadoc**.
2. Click the **com.edsplm.tc.req.databeans** package link.
3. From the Class Summary list, click the **DataBean** link.

The constants are defined in a HTML table.

The **Member Count** property returns an approximate value. Some members may not be visible because they are deleted or not effective. For performance reasons, these are not filtered out of the member count. To get a precise member count, use **Actual Member Count**.

EXAMPLES

```
set name [getValue $obj Name]
set txt [getValue $object Text]
```

If **Assign to** is a dynamic choice property that exists on the object.

- To obtain the value in the **Assign to** property, make a call as follows:

  ```
  getValue $object "Assign To"
  ```

- To obtain the LOID of the object which is stored as the value in the **Assign to** property, make a call as follows:

  ```
  getValue $object "LOID:Assign To"
  ```

Get the number of items in a folder:

- Get the approximate number of items in a folder

  ```
  set memberCount [getValue $folder "Member Count"]
  ```

- Get the precise number of items in a folder

  ```
  set memberCount [getValue $folder "Actual Member Count"]
  ```

importDocument

DESCRIPTION

Imports a Word file or other document type into Systems Architect/Requirements Management.

SYNTAX

```
importDocument owner filename docLocation [subtype] [filetype]
```

ARGUMENTS

- `owner`: OBJECT – The document owner.
- `filename`: STRING – Full path name and the MHTML or XML file name.
- `docLocation`: STRING – Path name of the original document.
- `subtype`: STRING – For MS_WORD and MS_EXCEL import file types, `subtype` identifies the type or subtype name to use for all imported objects. Default is the base requirement type.

  For AP233 import file type, the `subtype` argument should specify the property name to use as the unique identifier for objects to be updated. If no updates are desired, specify "".

  For all other import types, the `subtype` argument should be present; however, it is unused (specify "").
- `filetype`: STRING – The type of file to import. MS_WORD, STYLESHEET, MS_EXCEL, EXCEL_TEMPLATE, AP233, XML, XML_UPDATE, SCHEMA or PROJECT. The default is MS_WORD.

NOTES

The imported file types are:

- MS_WORD – Word file in MHTML format.
- STYLESHEET – Word file in MHTML format. Only the stylesheet section is imported. Owner must be a Stylesheet.
- MS_EXCEL – Excel file in MHTML or XML format.
- EXCEL_TEMPLATE – Excel file in MHTML format. Owner must be an Excel Template.
- AP233 – AP233 file in STEP format.
- XML – XML file with information about new objects to be created.
- XML_UPDATE — XML file with information to update existing objects (when the ID matches an existing object's LOID) or create new objects.
- SCHEMA – XML file of schema objects. Owner must be a project.
- PROJECT – XML file for an entire project. Specify a real project for the owner. A new project is created.

For more information about the Architect/Requirements XML format, see the *Systems Architect/Requirements Management Project Administrator's Manual*.

Imports done through this API are always carried out within the web server process, even if the Systems Architect/Requirements Management's **ExternalImportExport** setting has been configured to use an external process. This could result in performance or memory consumption issues within the web server process.

RETURNS

STRING–Folder LOID number.

moveObjects

DESCRIPTION

Moves the source objects to the specified destination.

SYNTAX

```
moveObjects sources destination
```

ARGUMENTS

- `sources`: OBJECT_LIST – Objects to move.
- `destination`: OBJECT – New owner of moved objects.

RETURNS

OBJECT_LIST–The moved objects.

EXAMPLES

```
moveObjects $requirements $folder
```

restoreFromTrashcan

DESCRIPTION

Restores deleted objects. Restored objects are no longer marked as deleted and are moved to the specified new owner or back to their original owner.

SYNTAX

```
restoreFromTrashcan objects [newOwner]
```

ARGUMENTS

- `objects`: OBJECT_LIST – Object containing desired value.
- `newOwner`: OBJECT – New owner for the restored objects, default is to restore to the original owner.

RETURNS

OBJECT_LIST–The restored objects.

EXAMPLES

```
set object [restoreFromTrashcan $deleteObject]
```

runActivator

DESCRIPTION

Runs the specified activator specified by Activator ID.

SYNTAX

```
runActivator name current selected
```

ARGUMENTS

- `name`: STRING – The name of the activator.
- `current`: OBJECT – Object to set as the current object in the activator
- `selected`: LIST – List of additional information to pass into the activator

RETURNS

STRING - the activator result or an error message if the activator failed.

runReport

DESCRIPTION

Searches the database for objects that match the specified criteria and output them in a report file.

SYNTAX

```
runReport  report template startingObject live
```

ARGUMENTS

- `report`: OBJECT – Saved Search object that contains the search query.
- `template`: OBJECT - Excel Template, Document Template or saved View object used to format the report output.
- `startingObject`: OBJECT - Starting object for the search. If omitted or empty then the starting object for the saved search is used.
- `live`: BOOLEAN - Pass **true** if exporting to Excel via a Template or View to get a "Live" spreadsheet, or **false** for static spreadsheet.

  The `live` argument is not used when exporting to MS Word. Export to MS Word is implied when the `template` argument identifies a Document Template object.

RETURNS

Full pathname of the server file where the Word or Excel report output is written. The **createAction FileDownload** API can be used to move this file to the client workstation.

EXAMPLES

```
set report "\\\\[getValue $currentProject Name]\\Reports and Formatting\\myReport"
  set report [getValue $report LOID]

  set template "\\\\[getValue $currentProject Name]\\Reports and
Formatting\\myTemplate"
  set template [getValue $template LOID]

  # Run the report, download the server file to the client and open in Excel
  set filePath [runReport $report $template $startingObject "false"]
  createAction FileDownload [list $filePath MS_EXCEL]
```

search

DESCRIPTION

Return a list of objects matching the search parameters.

SYNTAX

```
search searchSpec startingObject
```

ARGUMENTS

- `searchSpec`: STRING – XML specification for the search.
- `startingObject`: OBJECT – Starting point for search

RETURNS

OBJECT_LIST– Nested list of objects matching the search criteria.

NOTES

Constructing the `searchSpec` XML can be difficult. Instead of constructing it from within your Tcl, use the **Search** module user interface to construct and debug a search, save the search and let TcSE give you the XML. In the **Administration** module, select the **Search** object and scroll the **Properties** tab to find its **Script** property. Triple-click in the field and use Ctrl+C to copy the script's XML text to the clipboard. Now, paste the text in a text editor, or directly into your Tcl code. You can see where to substitute different property names or values in WHERE clauses, or conditionally exclude or add sections of the script as required.

EXAMPLES

```
set results [search $searchXml $obj]
```

search

DESCRIPTION

Searches the database for objects that match the specified criteria. The search is not case sensitive.

SYNTAX

```
search base nameSpec contentSpec types [caseSensitive]
```

ARGUMENTS

- `base`: OBJECT – Folder or project to search in.
- `nameSpec`: STRING – The object name search criterion.
- `contentSpec`: STRING – The body text search criterion.
- `types`: STRING – Object types to include in the result, see DataBean.TYPE.
- `caseSensitive`: BOOLEAN – True to perform case sensitive search, false for noncase sensitive search. The default is false.

RETURNS

OBJECT_LIST— Objects matching search criteria.

NOTES

Types may only inclueNoteDB, FolderDB, and RequirementDB. The wildcards ? and * may be used in the spec strings. **contentSpec** applies for notes and requirements; it is ignored for folders.

EXAMPLES

```
set reqs [search $project "" "shall" {RequirementDB}]
```

sendEmail

DESCRIPTION

TcL Command class to send E-Mail to List of EmailIds.

SYNTAX

```
sendEmail recipients subject objectContentFormat objects sendMailAs
mailBodyMessage docTemplate
```

ARGUMENTS

- `recipients`: LIST – List holding recipients IDs.
- `subject`: STRING – Variable to hold the subject of the mail.
- `objectContentFormat`: STRING – Variable to hold object's content format options: Text,HTML,MHTML or URL.
- `objects`: LIST – List of Systems Architect/Requirements Management objects.
- `sendMailAs`: STRING – String to hold send mail option: send as body/attachments.
- `mailBodyMessage`: STRING – String to hold mailBodyMessage.
- `docTemplate`: STRING – String document template used to get object's content as expected document.

RETURNS

- `mailSendStatus`: Boolean indicating success or failure of sendMail function.

NOTES

- The *recipients* argument is a Tcl list including one or a combination of the following:
    - Actual email addresses in the form name@host.
    - UserGroup object Name or LOID.
    - User object Name or LOID.
- When a User is identified by Name, LOID or User Group membership, the email is sent using the **Email** property value for that user.
- The *objects* argument must be a list of LOIDs.
- TcSE's MailServerIP configuration setting must point to a valid email server.

setEnvironment

DESCRIPTION

Sets one of Systems Architect/Requirements Management' configuration parameters.

SYNTAX

```
setEnvironment param value
```

ARGUMENTS

- `param`: STRING – Name of the parameter to set.
- `value`: STRING – The new value for param.

RETURNS

VOID.

NOTES

Parameters that may be set are:

- `ChangeList`: Value may be set to true or false. Setting `ChangeList` to false suppresses change handling in Systems Architect/Requirements Management. This can be useful to improve the performance of large Tcl transactions. Turning change handling off suppresses After activator execution, but Before activators run as usual. Also, turning change handling off results in the Systems Architect/Requirements Management client not refreshing completely.
- `checkRedundantLinks`: Value may be set to true or false. The default value is true. Setting `checkRedundantLinks` to false suppresses redundant relationship checking when creating trace links and group links. This can be useful to improve the performance of operations that create a large number of links. The setting lasts until it is explicitly changed again or the end of the transaction. Exercise caution when setting `checkRedundantLinks` to false as it creates redundant links.
- `checkCircularLinks`: Value may be set to true or false. The default value is true. Setting `checkCircularLinks` to false suppresses circular relationship checking when creating trace links and group links. This can be useful to improve the performance of operations that create a large number of links. The setting lasts until it is explicitly changed again or the end of the transaction. Exercise caution when setting `checkCircularLinks` to false as it creates circular links.
- `ContextProject`: Value passed must be the LOID of a project object. Systems Architect/Requirements Management always has the notion of a "current project". This value is reflected in the **currentProject** global Tcl variable. But, there are times when a different project context is needed. For example, the getValue call for the "User Access" property of a User object returns that person's access to the current project. This example would allow getting the access level for a different project:

  ```
  setEnvironment ContextProject $proj2
  set access [getValue $user "User Access"]
  ```

  The Project context change only applies during the activator execution where the `setEnvironment` call is made. When the activator ends, the ContextProject is reset to the current project. But, within the same activator, it may be necessary to reset the context project back to the original value as it could influence later API calls within that activator. If your activator continues on to perform other actions it is better to reset the context project to the current project:

  ```
  setEnvironment ContextProject $currentProject
  ```

- `FastMode`: Boolean option to enable or disable implicit re-number refresh. The default is true, which enables implicit re-number refresh. Requirements and building blocks have a hierarchy property. When a numbered object is created, deleted or moved it may cause some of its siblings and descendents to be re-numbered. Refreshing the client can be expensive when a large number of siblings are present. Suppressing re-number refresh may improve performance significantly, at the cost of displaying outdated number information until the next refresh.

  The `FastMode` setting is effective for the entire client session. The mode applies until the client logs out or it is reset with another `setEnvironment` call.

- `FlushUndo`: No value required. `FlushUndo` empties the undo queue so no prior transactions can be undone.

- `initProgress`: Macros and menu commands run from the client displays the client's progress meter thereby preventing timeouts. `initProgress` allows Tcl developer to inform users of a command's progress and enables cancelling the command. Tcl `setEnvironment` actions controls the content displayed in the progress dialog.

  ```
  setEnvironment initProgress <message> <totalSteps>
  ```

  The given message is displayed in the progress dialog's message area. The `totalSteps` given in the `initProgress` informs the progress meter of the "steps" it required to reach 100%. The `currentStep` values passed in subsequent `updateProgress` calls must be from 0 to the `totalSteps` value. If `totalSteps` is omitted, the progress meter oscillates and does not show a progression from 0% to 100%.

- `MessageQueue`: Value may be omitted or set to **clear**.
  - If the value is omitted, the current contents of the message queue (message tags, not the full text) is returned.
  - If the value is **clear**, the message queue is also cleared.

  API callers need to use this call with caution. Users may be left unaware of important messages when this call is used.

- `queueResult` – This command causes a copy of the results of the current server call to be stored off so they can be retrieved and processed by the TcSE client at a later time. This is useful in situations where a server call is made from something other than the TcSE client, such as a JSP, but you want the results to be handled in the client. Login activators run during a server call from the login JSP so this command is needed if the login activator generates any messages or actions that need to be processed on the client. Information on the result includes:
  - Change list – Information about objects modified in the transaction that is used to refresh the client.
  - Message list – Message generated by the TcSE server or the Tcl displayMessage command.
  - Action list – actions generated by the Tcl createAction command.

  If this command is included in Tcl code that was run during a call from the client it will be ignored. This prevents the same result information from being processed twice. Once a queued result is saved on the server it will be returned to the client immediately following the next server call from the client. This means that some user action, such as selecting an object, is required in order to see the result.

- `SaveData`: Allows Tcl to save values that can be retrieved during a different transaction later in a user's session. This data will be lost if the session times out, or when the user logs out. The form of the call to save a value is:

  ```
  setEnvironment SaveData uniqueID value
  ```

  The `value` argument is any Tcl string. The `uniqueID` argument is a string identifier, which will be used later to retrieve the value. If a value has already been saved with that ID, the old value is replaced by the new one. But, that old value is returned as the result of the setEnvironment call. So, you an retrieve the old value and set a new one all as one call:

  ```
  set oldValue [setEnvironment SaveData uniqueID newValue]
  ```

  The form of the call to just retrieve a previously saved value is:

  ```
  set myValue [setEnvironment SaveData uniqueID]
  ```

  If there is no previously saved value with that ID, the result is an empty string, with no error indication. A retrieved value remains stored and can be retrieved again. To release the memory from a saved value, set it to the empty string:

  ```
  setEnvironment SaveData uniqueID ""
  ```

  This call retrieves the existing value and clears it as one operation:

  ```
  set oldValue [setEnvironment SaveData uniqueID ""]
  ```

  Clearing values that are retrieved and no longer needed is a recommended practice. Values saved with this feature occupy Web server memory associated with the user's HTTP session. Use this feature for storing simple flags, or short values. Avoid saving large amounts of text this way.

- `SetResponse`: Value is **Yes** (the default value) or **No**. Use the `SetResponse` parameter for operations that require a user response before executing. For example, some database modifications cause the system to prompt you with a warning; you must click **Yes** to continue. When this occurs during Tcl execution, the programmer must provide the response using the `SetResponse` parameter.

  The `setEnvironment` method supports the following named responses:

`OPT_PARAM_DELETE_PROJECT`	Used by the `deleteObjects` method, for objects in the Administration module.
`OPT_PARAM_CREATE_BASELINE`	Used by the `createBaseline` method.
`OPT_PARAM_NO_PENDING_CHANGE_ CREATE_BASELINE`	Used by the `createBaseline` method.

  For example, a selected activator can be deleted by using a macro that contains:

  ```
  setEnvironment SetResponse Yes OPT_PARAM_DELETE_PROJECT
  deleteObjects $selected
  ```

  The unnamed response `""` is used by all other methods.

- `SessionID`: No value required. `SessionID` contains the value the server maintains for the current Systems Architect/Requirements Management `SessionID`. This value can be used for

automating Systems Architect/Requirements Management logins from JSP code launched from an activator or macro.

- `assignRoins`: In order to avoid locking conflicts with the ROIN counter object a ROIN is not assigned to a requirement when it is created. ROINs are instead assigned when the transaction is committed. If you need to examine the ROIN of a newly created requirement you will need to force the assignment of a ROIN. The `assignRoins` command will assign ROINs to all the requirements created in the current transaction that do not yet have a ROIN.

  ```
  setEnvironment assignRoins
  ```

- `updateProgress`: Macros and menu commands run from the client displays the client's progress meter thereby preventing timeouts. `updateProgress` allows Tcl developer to inform users of a command's progress and enables cancelling the command. Tcl `setEnvironment` actions controls the content displayed in the progress dialog.

  ```
  setEnvironment updateProgress <message> <currentStep>
  ```

  The given message is displayed in the progress dialog's message area. The `totalSteps` given in the `initProgress` informs the progress meter of the "steps" it required to reach 100%. The `currentStep` values passed in subsequent `updateProgress` calls must be from 0 to the `totalSteps` value. If `totalSteps` is omitted, the progress meter oscillates and does not show a progression from 0% to 100%.

- `deltaMode`: The web application parameter `DB.DeltaCapture` sets the capture settings for the LOIDs of modified database objects. The LOIDs of modified database objects are captured if `DB.DeltaCapture` is set to **true**. `DB.DeltaCapture` can be overridden within the current transaction, using the boolean `deltaMode` parameter.

  Following is the command to disable LOID capture:

  ```
  setEnvironment deltaMode false
  ```

- `includeImages`: For use only in activators used in object templates. An activator in an object template can return rich text content from a different Requirement or Note by referencing its HTML property. In that case images in the referenced object are not included. The `includeImages` keyword instructs the Word export process to include images for the specified list of objects. For example:

  ```
  setEnvironment includeImages [list $otherRequirement]
  return [getValue $otherRequirment HTML]
  ```

EXAMPLES

Example of common usage:

```
setEnvironment ChangeList false
```

Example of common usage of `SessionID` parameter:

```
set sessionID [setEnvironment SessionID]
createAction launchWebBrowser [list urlString sessionVarName $sessionID]
```

setObject

DESCRIPTION

Sets a list of property values on the given object.

SYNTAX

```
setObject object propertyList valueList
```

ARGUMENTS

- `object:` OBJECT – The Systems Architect/Requirements Management object.
- `propertyList:` STRING_LIST – List of desired properties, see DataBean.PROPERTY.
- `valueList:` STRING_LIST – List of the values to set.

RETURNS

VOID.

NOTES

In addition to the system properties defined in **Databean.PROPERTY**, any applicable user defined properties may be used.

If you are setting the value of the property by passing some String, simply use the regular `setObject`. But if you want to pass the LOID of the objects for the Pick list property as its values, you need to make the same `setObject` call with `LOID:` as a prefix for the property. If you are setting the value on a pick list, hat value should be a name of the object or its LOID.

For additional information, see Using A Pick List Activator in chapter **Unsatisfied xref number**, ***Unsatisfied xref title***.

For information on user-defined, dynamic choice property (pick list property, see the section *Setting a Dynamic Choice List for a Choice Property Definition* in the *Systems Architect/Requirements Management Project Administrator's Manual*.

Setting a dynamic choice property using the LOID: form is significantly faster than setting it by value.

EXAMPLES

```
setObject $Requirement {Name {Paragraph Number}} {Speed 2}]
```

If **Assign to** is a dynamic choice property that exists on the object, and **c User Group** is an object that exist in the project:

- To set the object of the **Assign To** property, make the following call:
  ```
  setObject $currentObject "Assign To" "c User Group"
  ```
- To pass the LOID of the object of the **Assign To** property as its values, make the following call:
  ```
  setObject $currentObject "LOID:Assign To" "497.0.7800
  ```

  497.0.7800 is an example LOID.

setPassword

DESCRIPTION

Changes a user’s password.

SYNTAX

```
setPassword name newPassword oldPassword
```

ARGUMENTS

- `name`: STRING – The name of the user.
- `newPassword`: STRING – The new password for user.
- `oldPassword`: STRING – User's old password.

RETURNS

OBJECT–The modified user object.

NOTES

May be used only by the project administrator or to change your own password. Project administrators do not need to specify `oldPassword` when changing another user's password.

EXAMPLES

```
setPassword "fred" "xyzzy" "plugh"]
```

setUserPreferences

DESCRIPTION

Allows users to specify their location so that information can be formatted appropriately.

SYNTAX

```
setUserPreferences names values
```

ARGUMENTS

- `name`: STRING_LIST – List of preferences names.
- `values`: STRING_LIST – List of preference values.

RETURNS

None.

NOTES

Preferences that can be set are:

> `timezone`: Specify the time zone of the client so dates and times are displayed in local time (see **DataBean.PROPERTY.TIMEZONE** and **java.util.TimeZone**). Timezone determines the time shift from GMT.
>
> `locale`: Specify the locale of the client (see **DataBean.PROPERTY.LOCALE** and **java.util.Locale**). Locale controls how date, time and numbers are formatted, and in what language months are displayed

Java developers can use `setObject` to set preferences.

EXAMPLES

```
SetUserPreferences timezone “America/Los_Angeles”

setUserPreferences [list timezone locale] [list “GMT-8:00” “en_US” ]
```

setValue

DESCRIPTION

Sets the value of a property on the given object.

SYNTAX

```
setValue object property value [lightweight]
```

ARGUMENTS

- `object`: OBJECT – The Systems Architect/Requirements Management object.
- `property`: STRING – The property to set, see DataBean.PROPERTY.
- `value`: STRING – New value for property.
- `lightweight`: BOOLEAN – True if setting a lightweight property, false otherwise. The default is false.

RETURNS

BOOLEAN – Value indicating whether or not the operation was successful.

NOTES

In addition to the system properties defined in **DataBean.PROPERTY**, any applicable user defined property may be used.

Lightweight properties allow an API caller to save a named value on an object without requiring a property definition. Setting a lightweight property creates, or updates, a text property instance with the given name and value. Setting a lightweight property to an empty string (“”) causes the lightweight property instance to be deleted.

Lightweight properties are intended for cases where only a minority of objects need that particular value saved. If most or all instances of a type require the value, an ordinary property definition should be defined and assigned to that object type. Lightweight properties have several limitations:

- Cannot be displayed as a column in the Systems Architect/Requirements Management client views
- Appear in the Properties tab, but are not editable
- Do not appear in the selection list of properties in the Search module
- Cannot have a lightweight property of the same name as a System or User Defined property on the same object.

When certain properties are set, such as the Properties property of a Type Definition, you are prompted with a warning message and must click **Yes** to continue. When setting these properties using `setValue`, the answer to the question must be supplied. The following line of code must be included before the setValue or the property will not be set:

```
setEnvironment SetResponse Yes
```

To add, remove, or reorder the allowed choices for a Choice List property definition, set its **Choice List** property. When calling setValue, you must pass a string that includes alternating new and old values, in the order you want them to be in. The values are separated by the internal delimiter (middle dot (**.**)) as follows:`new.old.new.old`. Added choices or deleted choices are indicated by a null string (empty string) between the delimiters. That is, when adding a choice, there is no old value, so there are two adjacent middle dot characters. Likewise, when deleting a choice, the new position is empty.

If you are setting the value of the property by passing some String, use the regular `setValue`. But if you want to pass the LOID of the objects for the Pick list property as its values, you need to make the same `setValue` call with `LOID:` as a prefix for the property. If you are setting the value on a pick list, that value should be a name of the object or its LOID. For more information, see Using A Pick List Activator in chapter **Unsatisfied xref number**, ***Unsatisfied xref title***.

Setting a dynamic choice property using the LOID: form is significantly faster than setting it by value. For information about setting a dynamic choice list for a choice property, see the *Systems Architect/Requirements Management Project Administrator's Manual*.

Use the **Static** property for freezing and unfreezing an object.

- Setting **Static** to **true** freezes the object.
- Setting **Static** to **false** unfreezes the object.

For example:

```
# freeze the requirement
setValue $req Static true
```

Due to delayed ROIN assignment it may be necessary to explicitly assign ROINs before setting the text content (MHTML) of a requirement (see `setEnvironment assignRoins`).

The MHTML keyword is used to retrieve the text content of a note or requirement including graphics. For performance reasons the OLE content is not included when the MHTML keyword is used. If the text content is opened in Microsoft Word, or used to set the content of another requirement, the OLE objects will be represented with a graphic, but the OLE application cannot be launched. To include OLE content use the MHTML_FULL keyword instead. The MHTML_FULL keyword is only supported in getValue, always use MHTML in setValue.

For example:

copy the content of a note to a new note including OLE objects

`setValue $`*`newNote`* `MHTML [getValue $`*`oldNote`* `MHTML_FULL]`

The list of Java API functions is provided in the API Javadoc. The API Javadoc describes each function along with the response expected from the server. Additionally, the list of property names, lists, keywords and other constants are defined in the **DataBean** class in API Javadoc.

You can access the Javadoc from the Systems Architect/Requirements Management home page.

1. Click **API Javadoc**.
2. Click the **com.edsplm.tc.req.databeans** package link.
3. From the Class Summary list, click the **DataBean** link.

The constants are defined in a HTML table.

EXAMPLES

```
setValue $Requirement Name "Maximum speed"
```

To make the changes in the following table:

Table 5-1. Sample Change List

New	Old	Description
Susan	Susie	Change name from Susie to Susan.
Mark	Mark	No change.
Sally	`null`	Add Sally in the third position.
`null`	Peter	Delete Peter.
Henry	`null`	Add Henry in the fourth position.
y	x	Rename from x to y; troublesome rename, server catches.
x	y	Rename from y to x; troublesome rename, server catches.

The Tcl code to set the choices in the example would be:

```
global choiceDelimiter
set choices [list Susan Susie Mark Mark Sally "" "" Peter Henry "" y x x y]
setValue $object "ChoicePropertyName" [join $choices $choiceDelimiter]
```

- To set the object of the **Assign To** property, make the following call:

  ```
  setValue $currentObject "Assign To" "c User Group"
  ```

- To pass the LOID of the object of the **Assign To** property as its values, make the following call:

  ```
  setValue $currentObject "LOID:Assign To" "383.0.19579"
  ```

  383.0.19579 is an example LOID.

uncoupleShortcuts

DESCRIPTION

Uncouples a shortcut from the master object, creating a copy of the master object. If the shortcut includes the children of the original object, then it is a deep copy. Otherwise only the master object is copied.

SYNTAX

```
uncoupleShortcuts shortcut
```

ARGUMENTS

- `shortcuts:` OBJECT_LIST – The shortcut objects to be uncoupled.

RETURNS

newObjects: The new object copies.

EXAMPLE

```
uncoupleShortcuts $shortcut
```

writeLog

DESCRIPTION

Writes a message to the web server's log file.

SYNTAX

```
writeLog message
```

ARGUMENTS

- `message`: STRING – Message to place in log file.

RETURNS

VOID.

NOTES

The name and default location of the web server log file varies with each of the web servers, and the server may be configured to use a different location.

Use caution in writing to the log file. Excessive logging can impact performance, and lead to very large log files. Recording error conditions or other significant events is appropriate, but do consider how frequently these events may happen in production. Debug printing is appropriate when developing activators or other Tcl, but must be removed for production operations.

Chapter 6: Using Activators

This chapter describes activators, including macros, and explains how to use them.

Introduction

In Systems Architect/Requirements Management customized behavior can be associated with object types. This behavior is defined in objects called activators. For example, an activator can be associated with requirements so that it is triggered when any requirement is modified.

Activators can access the Systems Architect/Requirements Management database, operating system commands, and other applications. Systems Architect/Requirements Management uses Tcl (Tool Command Language) as the language for programming activators.

Access Privileges for an Activator

Users need the script authoring privilege to create, edit and run activators in each project. To give this privilege to a user for a particular project, change the user's user object in the project's users folder by setting the Additional Privilege property to include Script Authoring. Because Enterprise users have access to all projects in the system, their user objects appear only in the users folder of the Systems Architect/Requirements Management Administration project. However, a user does not need script authoring privilege if an activator is triggered as a result of another user's actions.

You must have the following access privileges to create, edit, or view a Tcl script for an activator:

- To create or edit an activator, the maximum privilege level assigned to you in the project must be **Project Administrator** or **Enterprise Administrator**.
- After an activator is created, it can have a security profile that grants editing rights to other users with **Read and Write** access privilege.
- In all cases, you must have the special privilege of **Script Authoring** in the project to create, edit, or view the text of an activator.

For more information about project access and special privileges, see the *Systems Architect/Requirements Management Project Administrator's Manual*.

Describing Activator Objects

Activators are executable behavior that trigger and execute when certain events occur as a result of user action in the Systems Architect/Requirements Management client. Activators contain executable code written in Tcl. When an activator triggers, the Tcl code it contains is executed. For example, an activator that triggers on requirement modification could:

- Create a note to record the rationale for the requirement change.
- Set an attribute showing that the requirement had changed and needs review.
- Notify members of the Configuration Control Board of the change through E-mail.

Activators are created in the Activators folder of a project in the Administration module. Activators have an event property that indicates which events cause the activator to trigger.

To enhance performance and for technical reasons, there are cases where Activator events are not triggered by design.

- Activators triggered by a Before event do not cause other Before activators to be run. But, Before activators cause After activators to be run.
- Activators triggered by an After event do not cause any other Before or After activators to be run.
- After events are not triggered when the `setEnvironment changeList FALSE` option is in effect. The internal mechanism that collects the change list also drives the After events, so that information is not available when the change list is suppressed. Before activators are still run as usual.
- When you import Word, XML, AP-233, or Excel file in create or update mode, the **Undo** command does not work.
- When you update properties while importing an AP233 file, Change Logs are captured if you enable **Change Logging** for those modifications. For all imports (Word, XML, AP-233, or Excel) in create mode, changes should not be logged even if **Change Logging** is enabled

Creating Activators

This section presents the steps required to set up an activator.

See Access Privileges for an Activator for information about access privileges required to create an activator.

To set up an activator:

1. In the administration module, open a project, select the Activators folder and run the **New:Activator** command.
2. Open the activator and enter the Tcl script to execute.
3. Edit the Activators Events property and set the events that will trigger the activator.
4. Open the Type Definitions folder, select an object type, edit the activators property and select the activator so that it applies to objects of this type.

- After the Tcl script of an activator is saved, there may be a small delay of a few seconds until the changes are saved on the server. If the user reopens the **.tcl** file immediately, the changes may not be reflected.

  This delay is caused by the mechanism used to determine whether any changes are made to the file. The Systems Architect/Requirements Management has no direct access to the editing application as it allows the user to use any editor to edit an activator. Therefore, the Systems Architect/Requirements Management must poll the edited file to check whether there are any changes. Polling too often may affect the performance, so there is an interval of a few seconds between the polls.

- When you open an activator for editing in multi-pane editors, the Systems Architect/Requirements Management places a reservation on the activator. You must use the **Tools→Release Reservation** command if other users want to edit the same activator. For more information, see the chapter *Working With Object Properties* in the *Systems Architect/Requirements Management User's Manual*.

Using Activators in Excel and Object Templates

Excel templates and object templates allow activators to be called as follows:

- {%**Activator**%*Name*}

  The activator is called when exporting an object using the Excel template row or the object template. If an activator by the given name is found in the current project, it is used. If none is found, the Architect/Requirements Administration project is checked. If none is found there, an error is generated.

  When the activator is run, the object being exported is identified in the **currentObject** global Tcl variable. Arguments can be passed to the activator using this syntax:

  For more information about Excel templates and document templates, see the *Systems Architect/Requirements Management Project Administrator's Manual*.

- {%**Activator**%*Name*%*Arg1*%*Arg2*%*Arg-n*}

From within the activator code, the arguments are found as a Tcl list in the **selected** global Tcl variable.

Properties of the current object can be passed as argument values by embedding the {%*propName*} syntax, as it is normally used in Excel and object templates:

- {%**Activator**%*Name*%{%*Create User*}%{%*Change User*}}.

Within the activator, those arguments are retrieved from the Tcl list like this:

```
set createUser [lindex $selected 0]
set changeUser [lindex $selected 1]
```

When used within an object template, an activator is expected to return valid HTML text. This could be a plain text string with no HTML markup or the string could include simple HTML markup, such as `<B>bold</B>` and `<I>italic</I>`. It can also include complex tables, using the `<table>`, `<tr>`, `<td>` notation, or any other valid HTML content that Microsoft Word allows. If the string is to include any HTML reserved characters as visible text, they must be encoded using HTML conventions. For example, ampersand (&) is encoded as `"&amp"`, and less than (<) as `"&It"`. Non-ASCII characters must be encoded, else they do not display correctly. Architect/Requirements property values are allowed to contain non-ASCII characters, so strings returned by **getValue** must be encoded. Use the following command for encoding:

```
set encodedContent [java::call org.htmlparser.util.Translate encode $content]
```

When used in an Excel template, an activator returns one of the following:

- A static string to be displayed in that cell of the exported Excel spreadsheet. The returned string can be plain text, or can include HTML markup, as described above, and subject to what Excel allows in cell content. Even if the spreadsheet is exported to live Excel, this text is not associated with any object or property.

  The `getValue $obj` HTML return value for a requirement (or paragraph or note) may or may not be valid HTML to be inserted in an Excel cell. Excel does not support all the content and formatting that Word does, or in the same way. For example, paragraph breaks and certain style ranges in the HTML text cause extra rows to appear in Excel. These are traits of how Excel handles HTML content, and is not Architect/Requirements behavior. Also, if the text includes embedded graphics or OLE objects, they are not included in the result. If you want to include an objectâ€™s text, use the **Text** property, although it does not retain any markup.

- An object and property reference to create a live cell, as a colon-delimited string in the form *:nn.n.nnnn:propName:*, where *nn.n.nnnn* is the LOID of an object (in valid format), and *propName* is the name of a property of that object. In the Excel export file, the cell displays the current value of that property on that object. If the spreadsheet is exported to live Excel, cells exported in this form are associated with the object and property and can be edited in the same way as other live cells.

An object template activator can return rich text content from a different requirement or note by referencing its HTML property. When this is done the images from the referenced object are not included in the Word export. To include the images use the `setEnvironment includeImages` command. For example:

```
setEnvironment includeImages [list $otherRequirement]
return [getValue $otherRequirment HTML]
```

Defining Events

Systems Architect/Requirements Management recognizes many internal states, each of which could be considered an event. When such an event occurs, any activators defined for that event, on that type of object, are executed. Events generally fall into two categories:

- **Object Modify Events** occur when objects get modified. This includes relationships, moves, and editing properties.
- **Session Activity Events** are events that are triggered by logging into or out of Systems Architect/Requirements Management.

Objects modified while running an activator do not trigger an event.

Defining Object Modify Events

An Object Modify event type is generated when objects are modified, created, or deleted. There are two types of Object Modify events: Before Change events and After Change events.

- A *Before Change* event is generated just before an object is modified, but only after the necessary objects are locked and access control is checked.
- An *After Change* event is generated after the modification is complete, but just before the transaction completes and the response is returned to the user.

Before Change events are triggered only on objects that are selected for an operation, while After Change events are triggered for all modified objects. For example, if a folder is selected for deletion, a Before Delete event is triggered on the folder, but not on requirements in the folder. But After Delete events are triggered for all objects that are deleted. When more than one After Modify activator is triggered by an operation, they may be run in any order.

Before Change events are triggered when a modify operation is requested. After Change events are not triggered until the end of the API call, after all changes have been completed. Even if an object is modified multiple times, there is only one activator run for a particular event. For example, if two property values are modified, an After Modify Activator runs once for the modified object. The change list identifies which two properties were modified.

Before Change events are useful if you want to execute, check, or test a few conditions before the actual modification takes place. For example, to do additional consistency checking, enforce a design

methodology, or impose additional access control, you could define a Before Modify event on an activator.

After Modify events are useful if the user wants to perform additional operations on the objects involved in the event or on some other objects; for example, if you want to set a few attributes after an object is created.

The typical trigger for Object Modify events is a user modifying objects in the Systems Architect/Requirements Management client. But, Tcl API calls can also be the trigger. Ordinary activators running in response to object events do not cause other activators to be triggered. This behavior protects against the infinite recursion that's possible if one activator caused others to run. But, other Tcl code, running outside the context of an event, can cause Object Modify events to be queued and applicable activators to run. This includes Tcl running as macros and menu commands.

The following events are supported in Systems Architect/Requirements Management:

- **Before Create:** This event is set on the Type Definition for the type of object that is about to be created. But, the new instance of that type does not exist at the time of the Before event, so the target object of the activator is the intended owner of the new object. (You need two target objects when creating a link.) This activator could cause the Create action to be cancelled before any change takes place.
- **Create:** This event is generated after an object is created.
- **Before Modify:** This event is generated when a property of an object changes. The event is triggered before the actual database modification occurs.
- **After Modify:** This event is generated when a property of an object changes.
- **Before Delete:** This event is generated when an object is deleted. This operation occurs before the object is actually deleted. This event is triggered only on objects that are selected for deletion. Objects that are deleted as a side effect of deleting the selected object (such as notes, trace links, and children of the selected object) do not trigger this event.

  When a User Group object is deleted, a Before Delete event gets executed twice.

  - It is triggered first just after you reply **Yes** to the **Are you sure you want to delete the selected object?** message, just before the **This operation cannot be undone. Continue?** message appears. Even thought the Before Delete event has triggered at this point, the User Group is not yet deleted, and does not deleted if you reply **No**.
  - If you now reply **Yes**, then the Before Delete event is triggered the second time and the User Group is deleted.

  There is no way to tell the difference between these two Before Delete events. Unlike ordinary Groups, there is no After Delete event when a User Group is deleted.

  Performing actions in before activators generally does not work, and it is not recommended. For example, if you want to put deleted objects in a special folder instead of the trash can you cannot use a before activator. You can move the objects, but they are still deleted when the command continues.
- **After Delete:** This event is generated when an object is deleted. Because this event occurs after an object is deleted many operations, such as **setValue**, will fail. The object would first have to be restored from the trash can before it could be modified. After Delete events trigger for all objects that are deleted, not just the objects that are selected for deletion. For example, deleting a folder triggers After Delete activators on requirements in the folder.
- **Before Open:** This event occurs when text is opened for modification in Microsoft Word. **Before Open** activators can be used to prevent Word from opening or to open the text read-only. Use this Tcl code to open the text read-only:

  ```
  global actionContinue
  set actionContinue stop
  ```

Use the Tcl error command to stop the transaction and prevent Word from opening. If you wish to suppress the normal Tcl error message use this syntax:

```
error STOP
```

- **After Relation:** This event is generated when a relationship to an object changes. Relationship changes include moving an object and adding or removed a note, trace link, or child.

Defining Session Modify Events

Session Activity events are events that are triggered by logging into or out of Systems Architect/Requirements Management. The following Session Modify events are supported by Systems Architect/Requirements Management:

- **After Login:** This event is generated when you first login Systems Architect/Requirements Management. This event is also generated when a re-login occurs after session timeout.
- **Before Logout:** This event is generated before you log out of Systems Architect/Requirements Management Logout can occur either by exiting the application or from a timeout due to inactivity.

Using A Login/Logout Activator

The After Login and Before Logout events are a unique and you must consider that in writing activators that are enabled for these events.

The After Login event occurs in the server shortly after the user successfully enters their UserID/Password in the login Web page. This is before the client is initialized, and it occurs without the context of any target object, or even any project.

Similarly, the Before Logout event occurs in the server at a time when Systems Architect/Requirements Management client is already in the process of shutting down. This event also has no context of a target object or project, regardless of what object might have been selected in the Systems Architect/Requirements Management client user interface at the time.

As a result, these conditions apply to After Login and Before Logout activators:

- They must be defined in the Systems Architect/Requirements Management Administration project.
- They must be set as activators for the User type definition object in that project.
- Since the Systems Architect/Requirements Management client is either not yet initialized or is shutting down, calls to API functions that depend on the Systems Architect/Requirements Management client are ineffective, including `displayMessage` and `createAction`. For the **After Login** activators, this problem can be overcome by using the `queueResult` action of the `setEnvironment` API. However, this is not effective for **Before Logout** activators.
- **After Login** and **Before Logout** activators can examine, create, and modify objects. However, the errors occuring during these times are not reported to users because the client is not in a normal running state.
- Unlike other activators, errors occurring in **After Login** activators do not cause the current transaction to be aborted because that would abort the entire session login action. This abort means no one can log in. However, such errors may cause other side effects and hence, the **After Login** activators should be tested well before enabling that event.

Change Approval Routing Events

The following events are change approval routing events:

- **Change Submitted:** This event is generated when a change is submitted. It performs as the following tasks:
    - Prepares to send an email with present text and proposed text (new text). Addresses the email to everyone on Change Approvers list and on Change Notifiers list of the Change Approval object
    - Prepares the text messages for both sets of emails. Sends email to everyone on the Change Approvers list of the Change Approval object. If sending email is successful, it freezes the object (Requirements/Building Block).
    - Prepares and sends emails to everyone on the Change Notifiers list.
- **Change Approved:** This activator is generated by Change Response activator. This activator performs the following tasks:
    - Makes a line entry to the table in the Change Approval object with information such as time, user name, comment, and action.
    - Attaches the MHTML content of the Change Approval object and the content of the parent object (Requirement or the Building Block) to the email.
    - Freezes the parent object if it is not frozen.
    - Sends an email to each approver and to the originator of the change approval request.
- **Change Rejected:** This activator is generated by Change Response activator. This activator performs the following tasks:
    - Makes a line entry to the table in the Change Approval object with information such as time, user name, comment, and action.
    - Attaches the MHTML content of the Change Approval object and the content of the parent object (Requirement or the Building Block) to the email.
    - Freezes the parent object if it is not frozen.
    - Sends an email to the originator of the change approval request.
- **Change Response:** This activator obtains the user name (tcrUser), the user input text, and the user action, either approved or rejected, from the database and the jsp form input.

  This program enters the information such as user name, user comments, user's response, and the time in the table in the Change Approval object.

Import Events

An activator trigger on a folder can be used to prevent a document import. An after import event allows post processing of imported data.

- **Before import:** A before import allows you to cancel before a document import occurs. The type of import is passed in as a global. This code selectively prevents Word imports.

```
# Check if this is a Word import, could also be XML, AP233, PROJECT, SCHEMA,
# STYLESHEET, MS_EXCEL or EXCEL_TEMPLATE
if {$selected == "MS_WORD"} {

        # throw error to cancel transaction and provide message back to user
        error " Word import not allowed"
}
```

- **After import:** An after import event allows post processing of imported data. Only the selected object appears in the change list because change processing is turned off for performance reasons. A change flag indicates the type of import.

  Also because change list processing is disabled during import, Create activators are not triggered for most of the objects created by the import. Create events are only triggered for the topmost objects created, that is, those objects created directly in the folder where the import took place. Tcl can iterate from those objects down to visit all the objects created.

Storing Event Context

When an activator fires, information about the event that caused the activator to fire is passed into the Tcl environment. The event information is stored as Tcl global variables. The following global variables are set in the Tcl interpreter when activator execution begins.

- **currentObject:** The object that triggered the activator.
- **currentProject:** The project in which the **currentObject** resides.
- **event:** The type of event that triggered the activator.
- **tcrUser:** The user object of the current user.
- **tcrSubtype:** The subtype name for the object that is about to be created when a **Before Create** activator runs. The activator can reset this variable to a different subtype name, but it must be a subtype of the same base type.
- **selected:** A list of additional information that may be passed into activator when an activator is run using the **RequirementService runActivator** method.
- **changeList:** An array detailing exactly what changes were made to the **currentObject**. This list is not available in *Before* activators.
- **choiceDelimiter:** Separator used between the choices of a multi choice property.

Defining the Change List

Events, such as *After Modify*, do not indicate what was modified for an object. However, specific information about a change is kept in the change list. The change list array contains flags that indicate exactly what changes occurred. The change flags are the indexes of the Tcl global array named **changeList**. The presence of an index in the array indicates that type of change occurred. Some of the flags also have a value in the array. These values contain additional information about the change. For example, if a note is added to a requirement, an after relation activator fires. The `changeList` array has an *Add Note* entry whose value is the new note.

The following code detects that a note has been added and gets the note object:

```
set changeFlag “Add Note”
# get the list of change flags
set flags [array names changeList]
# check if  notes have been added
if {[lsearch $flags $changeFlag] != -1} {
    # note has been added, get the note objects
    set notes $changeList($changeFlag)
}
```

The change flag constants are documented in the Javadoc for the **AccessEnum** class.

Defining Flags

Three types of change flags can be set in activators.

Defining Relation Flags

Relation flags are the change flags that may be set in an *After Relation* activator. All the add and remove flags have a list of the added or removed objects as the **changeList** array value.

- **Add Member:** A child or member object has been added (either created or moved in).
- **Remove Member:** A child or member object has been removed (either deleted or moved out).
- **Add Note:** A note has been added.
- **Remove Note:** A note has been removed.
- **Add Defining:** A trace link has been created that gives the current object a new defining object.
- **Remove Defining:** A trace link has been deleted that removes defining object.
- **Add Complying:** A trace link has been created that gives the current object a new complying object.
- **Remove Complying:** A trace link has been deleted that removes a complying object.
- **Move Object:** The current object has been moved; the old owner is stored as the **changeList** value.
- **Modify Owner:** The current object has a new owner.
- **Restore Object:** The current object has been restored from the trash can.

Defining Modify Flags

Modify flags are the change flags that may be set in an *After Modify* activator. They indicate the properties that have changed and, in some cases, the original property value.

- **Modify Name**
- **Modify ROIN**
- **Modify Text:** The original text is stored as the array value.
- **Modify Owner**
- **Modify Value:** Some system property has been modified.
- **Modify Security:** The security profile property has been modified.
- **Modify Property:** A user-defined property has been modified; a list of the modified user property names is stored as the array value.

Defining Delete and Create Flags

Delete and create flags are set when an object is deleted or created.

- **Delete Object**
- **New Object**

Using A Pick List Activator

Pick list activators list the database objects that can be used in the choice list of a dynamic choice property. The pick list activator returns any list of objects, based on user-supplied logic. You can create any number of pick list activators in a project. For information about setting a dynamic choice list for a choice property definition, see the *Systems Architect/Requirements Management Project Administrator's Manual*.

Three sample scripts for pick list activators are in the **Activators** folder of the **TcSE Administration** project as example pick list activators. For more information, see the methods getObject, getValue, setObject, and setValue in chapter **Unsatisfied xref number**, ***Unsatisfied xref title***.

See Access Privileges for an Activator for information about access privileges required to create, edit, or view an activator.

- **Choice List:** This example returns a list of folders in a particular Project. This list can be used as a choice list in a dynamic choice property.

```
set project $currentProject
set members [getList $project FOLDER_LIST]
```

- **Users:** This example returns a list of users in the current project. This list can be used as a choice list in a dynamic choice property.

```
set project $currentProject
set members [getList $project USER_LIST]
```

- **User Groups:** This example returns a list of users and user groups in the current project.

```
set project $currentProject
set members [getList $project USER_AND_GROUP_LIST]
```

Pass Owner to Tcl Context

Depending on how they are used, Dynamic Choice Properties may have a choice list that is constant across all objects, or choices specific to an object and its state. For example, the choices for an Assigned To property might be the list of all Users in a Project. Or, the list of Assigned To choices might depend on a Status property. As objects move through their life cycle, the list of people the work might be assigned to would move from designers to engineers to QA to production. Architect/Requirements supports both cases.

The Choice Property's *Pass Owner to Tcl Context* indicates which behavior applies; **No** when the property always has the same list of choices, or **Yes** when the choices may vary depending on the object. Setting this value to **No** allows Architect/Requirements to operate more efficiently by caching choice lists. Setting it to **Yes** requires the activator to be run every time a choice list is needed.

When *Pass Owner to Tcl Context* is set to **Yes**, the `currentObject Tcl` global variable is set to the object of interest whenever the choice activator is called.

When A Pick List Activator Gets Called

This section presents the different actions that cause the activator to be called, and the `currentObject` value for each case; either an object that has the property, the Property Definition, or the Project. In the latter two cases, the activator author must return a choice list that's suitable for all objects.

- On a Dynamic Choice Property (e.g. **Assign To**) when you select a Pick List activator in the **Update Choice list Dynamically** field, Pick List activator is called, and the result gets populated in the **Choice List** field.

  currentObject = Choice Definition

- In the User module, when a **Dynamic Choice** property is opened for edit (the choice selection dialog is launched) the Pick list activator for the **Dynamic Choice** property is executed.

  currentObject = Selected Object in the User Module

- In the User module when you try to set the value on the Object for this Dynamic Choice property:

  currentObject = Selected object in the User Module

- In the Search module, selecting the Dynamic Choice Property itself:

  currentObject = Current Project

- If a Dynamic choice property is used in a macro's Form Values, and that macro got executed, when user double clicks the **Value**, the Pick List Activator is executed

  currentObject = Current Project

- In a Visio diagram when you edit Systems Architect/Requirements Management properties, when the Dynamic Choice property is edited for setting values, the Pick List Activator is executed.

  currentObject = Object in Systems Architect/Requirements Management User module for which property is edited in the Visio diagram

- On exporting the Systems Architect/Requirements Management object(s) which has the Dynamic Choice Property to Excel Live:

  currentObject = Current Project

  Choice lists are not dynamic in live Excel. The Choice List activator is run once at the time of Excel export to generate one list of choices that is used for all objects with a value for that property.

- When value is set for Dynamic Choice Property on Systems Architect/Requirements Management object in Excel Live:

  currentObject = Current Object for which value is set

- In XML Import and Excel Import, the Pick List activator is called once for each imported object that has a value to be set for a Dynamic Choice Property.

It is clear that Pick List activators can be called in many situations. It is the activator author's responsibility to insure that the activator will return a choice list that's valid for the currentObject in each case.

Pick List Activators and Performance

Activator authors must be aware that Pick List activators can have a significant negative impact on system performance. Unless the pick list activator author takes responsibility for testing the performance impact, it becomes a hidden burden on the system as a whole. The pick list overhead can impact many operations, but there is no way for a user to know the cause of the slow performance.

The degree of pick list impact on performance is influenced by:

- How often the activator is invoked
  - The number of object types that use the pick list property
  - How often instances of those types are encountered during user sessions
  - Whether the property definition has Pass Owner to Tcl Context set or not
- The work that the activator does to generate its list
  - Whether it just follows existing relationships, or uses search
  - How many objects it handles while generating its result list
  - Whether it uses transaction or session caching strategies
  - The overall quality and efficiency of the Tcl logic

The easiest way for activator developers to find out how often a pick list activator is being called is to add a **displayMessage** call in the activator. It must be commented out for production use, but this can be a very useful diagnostic tool during pick list activator development. In addition to testing normal user browsing and object actions, also test multi-object cases such as copying a structure, and document, Excel and XML import.

The number of types and objects using pick list activators depends on the business needs. In planning the schema for a project, avoid using pick list properties unless there is a legitimate need for them. Also, avoid using them on a base type, if the property is really only needed on a limited set of subtypes.

If **Pass Owner to Tcl Context** can be set to **No**, the system can avoid running the pick list activator in some cases. This helps performance, but is not a cure-all. The other techniques listed here must also be considered.

If at all possible, search must be avoided in collecting the list of objects in pick list activators. Caching of the list is especially beneficial when it’s necessary to use search to generate the list. If search is truly necessary, the search must be developed using the search module, and timed with actual production data. Different approaches for achieving the same search results can have dramatically different execution times.

Whether search is used or not, several different caching strategies can be beneficial. The appropriate caching approach depends on how dynamic the pick list choices are.

If the set of objects in a pick list seldom changes, then the best performance comes from calculating the list the first time it is needed in a session, and then caching it for the duration of the user session. The list can be saved using the **SaveData** option of the **setEnvironment** API. Each time the pick list activator is called, attempt to retrieve the list by making a **setEnvironment SaveData** call. If a list is returned, use it. Otherwise, calculate the list, save it for later reuse with a **setEnvironment SaveData** call, and return it for this use.

Often a pick list is based on some relatively static set of objects, but the exact subset returned by the activator may need to vary, depending on the state of the object the property applies to. In that case, use the setEnvironment SaveData approach to save the basic list. Then, each call to the activator would just filter that list down to the proper subset for that case.

If a pick list must be more dynamic, and always be sensitive to recent database changes, there is still some benefit to caching the choice list even within a single transaction. That's especially true if there is more than one choice property that uses the same dynamic list, or a closely related dynamic list based on the same underlying set of objects. For these cases, cache the list in a Tcl global variable, rather than using setEnvironment SaveData. Given that difference, the other comments in the two paragraphs above still apply.

Implementing Transaction Control

Both Before and After activators run within the same database transaction as the operation that triggered the event. If an uncaught error occurs in the Tcl, the transaction is rolled back and an error message is generated. To prevent the operation from occurring, you can throw a Tcl using the Tcl **error** command or set a global variable indicating the transaction should be aborted: set **actionContinue stop**.

The difference between these two methods is that the error command generates an error message while setting the global variable does not.

Setting **actionContinue** to **stop** does not immediately abort the Tcl execution. The flow of that activator continues. When the Tcl ends or returns, the Architect/Requirements server notices that the **actionContinue** flag is set. At that point, the current action is stopped and the transaction is aborted.

Creating a Tcl Where Clause

You can create a Tcl where clause to reference an activator to determine if an object meets the specified criteria or not. A Tcl Where clause can be used anywhere a where clause can be used.

Creating a Where Clause

To create the where clause:

1. Add a row to search criteria and indicate the command is a Tcl where clause.

   Systems Architect/Requirements Management presents a list of activators. The activator must return a Boolean value. Only activators of subtype **Where Clause** appear in the list. Activators are gathered from both the currect project and the administration project. Where clause activators in the administration project are available for use in all projects.

2. Select the activator to use.

   Systems Architect/Requirements Management displays usage description. The description is a text property on activator.

3. Optionally, enter parameters to pass into activator.
4. Systems Architect/Requirements Management adds Tcl where clause to query.

Searching With a Tcl Where Clause

To run search with a Tcl where clause:

1. Click the **Search** button.
2. Systems Architect/Requirements Management runs search query and displays result.

   Tcl where clauses are processed like other where clauses. The activator is run with the current object set as the object being evaluated. User-entered parameters are passed in as a global variable name parameter.

Tcl Where Clause Example

To verify that all functional requirements are met, you must ensure that all functional requirements are linked to a design element in the functional decomposition. This is done by creating a search called **Unallocated Requirements** that selects functional requirements that do not have complying objects. The result is a list of functional requirements that still need to be linked to the design. This search is inadequate to verify that the requirements are linked to elements in the functional decomposition as the **Complying Objects Count** property displays only complying objects. A Tcl where clause is required to examine the complying objects and verify that at least one functional design element (a building block with a **Functional Design** subtype) is included.

Unallocated requirement report script does not verify if the allocation is to the functional design. For example,

```
SELECT Functional Requirement
           WHERE Complying Objects Count = 0
```

Unallocated requirement report script with Tcl where clause verifies if the allocation is to a specific type. For example,

```
SELECT Functional Requirement
           WHERE hasComplying Functional Design
```

Tcl script for the **hasComplying** where clause activator. For example,

```
proc hasComplying {type} {
              foreach obj [getList $currentObject COMPLYING_OBJECT_LIST] {
                      if {[getValue $obj Subtype] == $type} {
                              return true
                      }
             }
             return false
}
hasComplying $parameter
```

This activator can be used in a where clause to prevent an object from being added to the results more than once in a **FOREACH ADD DEEP** command.

```
# initialize list to keep track of objects already in result
global visitedList
if { ! [array exists visitedList] } {
     set visitedList(0) 0
}
# check if the the current object is already in the list
set loid $currentObject
set newObject 1
catch {
  if { $visitedList($loid) == 1 } {
     # Found object in the list
     set newObject 0
    }
}
# add current object to visited list
set visitedList($loid) 1
# return result, 0 if this object was already in the visited list, 1 if the object
was
#not in the visited list
set newObject
```

Tcl Global Variables in Where Clause Activators

When a where clause activator is run during a search, the following Tcl global variables are set:

- **currentObject** is the object that is examined by the where clause for possible inclusion in the search result.
- **superior** is the object that the current object is indented under in the search result.
- **relation** names the relation that caused the superior object to be in the search results. For example, if the superior was added to the search results by an ADD Members statement, the relation value is **MEMBER_LIST**.
- **level** is the indention level of the superior object, where the left-most or top-level objects are at level 0. If the current object is added to the report, it is added one level above the level specified by **level**.
- **parameter** is a string holding the text that is entered in the **Parameters** field when adding the activator reference to the search script.

If a script's initial SELECT statement has a where clause with an activator, there is no superior object in this case; hence the values of **superior**, **relation**, and **level** are as listed here:

- **superior** has the same value as **currentObject**
- **relation** is empty
- **level** is **0**

As is customary with Tcl global variables, if referenced from within a Tcl procedure, these variables must be declared as global:

```
global currentObject superior relation level parameter
```

Writing Activators When Objects are Deleted, Restored, or Discarded

Writing activators that behave correctly can be difficult when objects are deleted to the Recycle Bin, restored from the Recycle Bin, or discarded from the Recycle Bin.

When an object is initially deleted, the possible activator events are:

- Before Delete on the specific object(s) being deleted
- After Delete on that object, plus all descendents and attachments deep
- After Relation (defining/complying) on any of the objects that had trace links to objects that weren't deleted

If an object is restored from the Recycle Bin, possible activator events are:

- After Relation (modify owner) on the one object that was restored
- After Relation (defining/complying) on any of the objects that had trace links to be restored

When an object is discarded from the Recycle Bin, either individually or by emptying the Recycle Bin, there are no possible events.

You are not allowed to modify objects in the Recycle Bin. So, other than the restore action, there should not be modification events on objects once they are in the Recycle Bin.

It is the responsibility of the activator author to monitor all cases where the activator may fire. Or, at least catch errors when they occur. Pay close attention to your After Relation events, since they can fire as objects go into the Recycle Bin and when they are restored from it.

The easiest way to tell if an object is in the Recycle Bin is to test the **Date Deleted** property:

```
if {[getValue $obj "Date Deleted"] == "" } {
      ...not deleted...} else {...is deleted...}
```

The test works for objects visible in the Recycle Bin, and all their descendents and attachments as well.

TC_XML Export Activator

When you export data to TC_XML, an activator is triggered. The activator configures the data being exported. The activator is triggered regardless of how you initiate the export. You can initiate the export in one of the following methods:

- Using the Architect/Requirements client
- Using the command prompt
- Using the Tcl **exportDocument** command

You can locate the activator by its name. Typically, the name of activator is **TcXmlConfig**. Architect/Requirements first searches for the activator in the current project. If a **TcXmlConfig** activator is not found then Architect/Requirements searches the administration project. This activator is for configuring TC_XML export. You can configure the activator to:

- Exclude unwanted data from the export
- Adjust type and property names
- Assign Teamcenter item IDs
- Specify the versions to export

The following sections provides information on the variables used in configuring the activator.

Assigning Teamcenter Item IDs

Item ID is the unique identifier assigned to Teamcenter objects. It is similar to a **ROIN** in Architect/Requirements. Teamcenter requires item IDs to be specified for TC_XML imports. You can specify the item ID in different methods. The method used is specified in the **itemIdRule** variable. You can use one of the following values for the **itemIdRule** variable to specify the method to be used:

- File
- Counter
- **ROIN**

Item IDs can be generated by Teamcenter and written to a file. Siemens PLM Software recommends using this method as it guarantees that the item IDs are unique and comply with Teamcenter naming conventions. To use the item IDs from a file, set the **itemIdRule** variable to **File** and specify the name of the file to use for which object type is specified in the **ItemIdFiles** map. Following is an example of assigning Teamcenter Item IDs using the file method:

```
set itemIdRule File
 set ItemIdFiles {
    Requirement {C:/temp/reqIds.txt}
    Paragraph {C:/temp/paraIds.txt}
    Note {C:/temp/noteIds.txt}
    Folder {C:/temp/folderIds.txt}
 }
```

The concept of inheritance applies when assigning Teamcenter Item IDs. If a requirement subtype name is not found in the map, Architect/Requirements searches the parent type.

Architect/Requirements can also generate item IDs. To do so, specify the item ID prefixes in the **itemIdPrefixes** list. Additionally, specify the counter format and starting number using the **itemIdStart** variable. Following is an example of assigning Teamcenter Item IDs using the Counter method:

```
set itemIdRule Counter
set itemIdPrefixes {
   Requirement "REQ-"
   Paragraph "REQ-"
   Note "CSTMNOTE-"
   Folder "REQSPEC-"
}
set itemIdStart 000100
```

You can also use the Architect/Requirements **ROIN** as the item ID in Teamcenter. To use the Architect/Requirements **ROIN** as the item ID, set **itemIdRule** to **ROIN**. If the **ROIN** is not available, the **LOID** of the type is used. You must ensure that the naming rules for your types in Teamcenter allow the Architect/Requirements **ROIN** format. Following is an example of assigning Teamcenter Item IDs using the **ROIN** method:

```
set itemIdRule ROIN
set itemIdPrefixes {
  Note "CSTMNOTE-"
  Folder "REQSPEC-"
}
```

Excluding data from export

Removing unwanted information from the export can simplify mapping and make the migration process faster. You can specify the object types and properties that are not required, using the following variables:

- **excludeTypes**

  Specifies the object types to be excluded. Unsupported types are automatically excluded.

- **excludeSysProps**

  By default, system properties that are unlikely to be useful in Teamcenter are excluded. Setting this variable replaces the default list. Hence, you have complete control over the system properties that are exported.

- **excludeProps**

  Additional list of user defined or system properties to be excluded.

Following is an example of removing unwanted information from the export:

```
set excludeTypes [list {Change Log} ImageNote DataDictionary]
 set excludeProps [list {Change Time} {Change User} {My Property}]
```

Adjusting type and property names

It is likely that the type and property names in Architect/Requirements are not an exact match of the corresponding names in Teamcenter. Teamcenter requires prefixes for custom type and property names while Architect/Requirements does not have such requirements. A mapping step is provided in the migration process to handle such mismatch. However, you can perform some of the mapping during the export process. You can add prefixes to type and property names which simplifies the mapping process.

You can also set up maps to specify the names to use in the export. You can use the following variables to set up the mapping:

- **typePrefix**

  Specifies the prefix for all object types.

- **sysPropPrefix** and **userPropPrefix**

  Specifies the prefix for system and user defined property names.

- **typeMap**

  Establishes a map for the name to use for a given object type.

  By default, **typeMap** maps the out of the box Architect/Requirements types to the corresponding out of the box Teamcenter type.

- **propMap**

  Establishes a map for the name to use for a given property. If a type or property name is included in a map, no prefix is added to it.

Following is an example of mapping the type and property names:

```
set typePrefix "Tcr4"
 set sysPropPrefix "Sys4"
 set userPropPrefix "Tcr4"

 set typeMap {
    Requirement "Tcr5MyReqType"
    DerivedRequirement "Tcr4DerivedReq"
 }

 set propMap(Project) "Tcr4TcSEProject"
```

Specifying the versions to export

You can control the requirement versions that are exported using the **versionRule** variable. Following are the valid values for **versionRule**:

- **Effective**

  Exports the currently effective version.

- **Baseline**

  Exports the current version and all baselined versions.

- **All**

  Exports all versions.

By default, only the currently effective version is exported. If multiple versions are exported, only the current version is configured into a requirement specification during the import. For baseline exports, snapshot objects named after the baseline are created to assist in configuration of the baselines.

Following is an example of specifying the version to export:

```
set versionRule Baseline
```

LOID Property

LOID is the unique database identifier for Architect/Requirements objects. Mapping **LOID** to a property in Teamcenter is useful for connecting Teamcenter objects to the corresponding Architect/Requirements object. The mapping is required for setting rich text content. The **loidProperty** variable contains the name of the Teamcenter property that stores the **LOID** value.

Following is an example of setting the **LOID** property:

```
set loidProperty Tcr4LOID
```

Activator Examples

This section provides examples of activators. You can copy and paste the examples in this section and use them directly. This is possible because the examples here are dependent on the currentObject.

Determining Requirements With The Same Name

The following example (figure 6-1) shows how many requirements have the same name in the present project- Activator Name **Same Name Requirements**. You set the Events property of the activator to **After Modify**. Then, enter this program in an activator and set the activator property of the Requirement Type Definition to **Same Name Requirements**.

```
proc Iterator { x name} {
global item
  set members [getList $x MEMBER_LIST]
     foreach z $members {
       set KK [getValue $z "Type Name"]
        if  {$KK == "Requirement" } {
             if  { [getValue $z Name] == $name} {
                      set item [expr $item+1]
             }
        }
        Iterator   $z $name

      }
    return $item
}

set item 0
set strName [getValue $currentObject Name]
set subMembers [Iterator  $currentProject $strName]
displayMessage " $item Requirements have the same name."
```

Figure 6-1. Activator Name, Same Name Requirements

Creating a Note

The following example (figure 6-2) creates a note before a requirement is opened for editing and copies its text property to the note.

```
set strTypeName [getValue $currentObject "Type Name"]
if {$strTypeName == "Requirement" } {
    #Creates a note
    set noteLOID [createObject oldText $currentObject "Note"]
    #Gets Text property of current object
    set strText [getValue $currentObject "Text"]
    #set text property of the note.
    setValue $noteLOID Text $strText
}
```

Figure 6-2. Creating a Note

Creating An After Delete Activator Event

The following activator example (figure 6-3), when set to activate After Delete for a requirement, moves the deleted requirement to a folder named **DeletedRequirements** if this folder exists or creates this folder if it does not exist.

```
# locate or create the deleted requirements folder
proc getDeletedFolder {} {
      global currentProject
      set folders [getList $currentProject MEMBER_LIST]
      foreach folder $folders {
            if {[getValue $folder Name] == "Deleted Requirements"} {
                  return $folder
            }
      }
      set folder [createObject "Deleted Requirements" $currentProject Folder]
      return $folder
}

# do not move children of the deleted requirement
if {[getValue [getValue $currentObject Owner] "Type Name"] == "TrashCan"} {

      # locate or create a folder for deleted requirements
      set folder [getDeletedFolder]

      # Undelete the requirement and place in folder
      restoreFromTrashcan $currentObject $folder

      # Change to deleted subtype so it can be filtered from search
      setValue $currentObject Subtype Deleted

      displayMessage "'[getValue $currentObject Name]' has been
         move to deleted objects folder"
}
```

Figure 6-3. After Delete Activator Event

Using createAction FileDownload

This section provides several examples using createAction FileDownload to download a file from the server to the Systems Architect/Requirements Management client workstation, and optionally open it in an application.

Producing an HTML File

The following example (figure 6-4) shows how to produce an HTML file from a report and launch a Web browser to view the report results.

```
set serverFile [runReport $report $template $startingObject $live]
   #run the report using parameters
set application Default #set IE as output application
createAction FileDownload [list $serverFile $application]
   #download the exported file from server and open using IE
```

Figure 6-4. Producing an HTML File

This example cannot be copied and pasted directly because it depends on the parameters, reportID, templateID, startingObjectID and Boolean live flag.

E-mailing Requirements or Producing PowerPoint Project from Requirements

The following example (figure 6-5) can be used to produce a PowerPoint project, one slide per Requirement from a selected folder. It can also be used to select a set of Requirements and E-mail them using Microsoft Outlook.

The output is determined by the application you select in the chooser. If you select PowerPoint, the downloaded file is opened in PowerPoint. If you select Outlook, a new E-mail window opens, and the downloaded file is attached.

```
set objectList $currentObject #take current object
set serverFile [exportDocument $objectList MS_WORD] #export this object to word
set application Chooser #choose the application
createAction FileDownload [list $serverFile $application]
   #download the exported file from server and open using selected application
```

Figure 6-5. E-mailing Requirements or Producing PowerPoint Project from Requirements

Using createAction RunJava

You can create custom Tcl code and custom Java code that run on the client JVM. You can use the custom code to interface with other applications or create custom interfaces such as a Java form for user input. **createAction RunJava** allows external programs to interface with Architect/Requirements using the C# interface. It allows the external Java code to be run from within Architect/Requirements through the Tcl code. It also allows identification of Java classes and execution of methods in the classes from the Tcl code of activators, macros, or custom menu items.

Setting up to run Java code in Architect/Requirements client

The location of the .**jar** file must be set by the administrator using the **Package.Location Web Application Configuration** parameter. The **jar** file must contain a method named **connectTcSE** with the prescribed method signature.

To prepare the Java code to run in the Architect/Requirements client:

1. Create a file named **TestRunJavaClass.java** containing the source code.
2. Compile the Java file. The class path must contain the three jar files from the Architect/Requirements client installation directory:
   - **tcrPackages**
   - **tcrShared**
   - **rcf**

   The following code is an example of the steps required to compile the Java file:

   ```
   set TCSE _DIR=C:\Program
   Files\SiemensPLM\Teamcenter\SystemsEngineering\Release_9.1

   set CP=%TCSE _DIR%\tcrPackages.jar;%TCSE _DIR%\tcrShared.jar;%TCSE
   _DIR%\rcf.jar

   javac -cp %CP% TestRunJavaClass.java
   ```
3. Add the compiled class to a jar file.

   ```
   jar -cf test.jar TestRunJavaClass.class
   ```
4. Update the **Package.Location** Web Application Parameter to reference the jar file path.

   Example:

   C:\test\test.jar

The ClientJavaAPI class

The **ClientJavaAPI** class is the beginning of a client interface.

To call an Architect/Requirements macro from the external Java code, the class file must import the **ClientJavaAPI** class found in the **tcrPackages.jar** file. The **tcrPackages.jar** file can be added to the **classpath** variable. You can also unzip it and use the **ClientJavaAPI** class file in the development area.

runMacro() Method

The **runMacro()** method of the **ClientJavaAPI** class is used to call a macro from the external Java code.

A basic method calls the macro by name by passing a space delimited string of LOIDs for the macro. For example:

```
public String runMacro(String macroName, String objectIDs)
```

Another method uses the macro name and the macro LOID along with the object IDs. In this case, if the macroID is passed and is not null, the macroName is not used. For example:

```
public String runMacro(String macroName, String macroID, String objectIDs)
```

Another extensively used method uses values passed into the macro from the form mechanism, passing in the form properties and the associated values. If the macroID is passed and is not null, then the macroName is not used. For example:

```
public String runMacro(String macroName, String macroID, String objectIDs, String[]
formProps, String[] formValues)
```

getTempDir() Method

If required, you can get a temporary directory from Architect/Requirements. The **getTempDir()** method returns a string containing the system temporary directory used by Architect/Requirements.

The **ClientJavaAPI** class must be available to the custom package. The **ClientJavaAPI** class must be imported with the following statement:

```
import com.edsplm.tc.req.client.shared.api.ClientJavaAPI;
```

It must be instantiated with the following statement:

```
protected ClientJavaAPI javaAPI = new ClientJavaAPI();
```

After instantiating the **ClientJavaAPI** class, you can use it through the following code:

```
result = javaAPI.runMacro(macroName, objectIDs);
```

OR

```
result = javaAPI.runMacro(macroName, null, macroParameters, formProps, formValues);
```

Java Development Environment

It is important to set up everything correctly, extract the files to the proper location, get the compiler setup to compile correctly and have it compile in the right location so that Architect/Requirements can find it.

Siemens PLM Software recommends creating subclasses of the delivered classes so that any potential changes made are not overwritten when a new Architect/Requirements deployment is made.

Complete the following tasks before you begin the customization:

- Set up the development environment, preferably with an IDE. You can use the Eclipse IDE from www.eclipse.org.
- Setup **Classpath** correctly.
- Set up packages correctly.
- Prepare to package the code in a **jar** file for distribution.
- Understand how to extend the delivered class to create new customized behavior.

The goal is to create a **jar** file for distribution or to replace the **tcrPackages.jar** file. You can start with unzipping the **tcrPackages.zip** file that contains the two external Java files in the working directory.

Here are additional recommendations to create the **jar** file.

- The new custom Java code must be in its own package.
- The Java package does not require a **main()** method; it is a collection of methods.

- The methods must be called from Architect/Requirements to be an extension of the Architect/Requirements client.

Unzip the **tcrPackages.zip** file and become familiar with the Java source code for the **TcSECMInterface.java**. It is the best example of the customization toolkit.

Code example

This section shows the source code for a complete Java program related to the examples in createAction RunJava in chapter **Unsatisfied xref reference**, ***Unsatisfied xref title***.

```
/* ---------------------------------------------------------------------------
 * This software and related documentation are proprietary to
 * UGS Corp.
 * (c) 2007 UGS Corp. All rights reserved.
 * ---------------------------------------------------------------------------
 */
import javax.swing.*;
import com.edsplm.tc.req.client.shared.api.ClientJavaAPI;

/**
 * This is a test class, to be used to test and understand the RunJava createAction.
 */
public class TestRunJavaClass
{
       ClientJavaAPI javaAPI = new ClientJavaAPI();

    // Can be called via RunJava - main method is not required, but may be used
    public static void main (String args[]) {
        printHelloWorld(args);
        printArgs(args);
    }
       // Can be called via RunJava - Run a macro on the server
       public void runTestMacro(String[] theArgs) {

              // specify macro name
              String macroName = "HelloWorldMacro";

              // run the macro and capture its return value, pass it the first
              // argument from the CreateAction RunJava call
              String result = javaAPI.runMacro(macroName, theArgs[0]);

              // display the return value
              showDialog(result);
       }

    // Can be called via RunJava - static method can be called
    public static void printHelloWorld(String someArgs[]) {
        // nothing is done with someArgs[]
        printIt("printHelloWorld");
        // popup dialog for user to click
        showDialog("Hello World!");
    }

    public void printAllArgs(String[] theArgs) {
        // this is the public method to show all arguments
        // it calls the private method that does the work
        printArgs(theArgs);
    }

    // Can be called via RunJava - no return, no parameters
    public void doNothing() {
        printIt("doNothing");
        // popup dialog for user to click
```

```
        showDialog("doNothing");
    }
```

```
    // Can be called via RunJava - prints first parameter sent, returns nothing
    public void doNothing(String[] someArgs) {
        printIt("doNothing(String[])");
        System.out.println(" > someArgs[0] is " + someArgs[0]);
    }

    // Can be called via RunJava -
    public String doNothingString() {
        printIt("doNothingString");
        // prints this line to the log
        return "prints this line to the log";
    }

    // Can be called via RunJava -
    public String[] doReturnStringArray() {
        printIt("doReturnStringArray");
        // prints these values to the log
        String[] returnValues = new String[]{"returnMsg1", "returnMsg2",
"returnMsg3"};
        return returnValues;
    }

    // Cannot be called directly via RunJava - must have this method!
    // This method is called "automatically" every time another method is called.
    // The intent and purpose of this method is to provide the necessary parameters
needed
    // in order for the external application to connect back with TcSE. To connect
back to
    // the TcSE client, the first three parameters may be used; to connect back to the
    // TcSE server, the last may be used.
    public String connectTcSE(String controllerPath, String serverIP, String
socketServicePort,
                              String sessionID) {
        System.out.println("here in connectTcSE, values are: " + controllerPath + ", "
                           + serverIP + ", " + socketServicePort + ", " + sessionID);
        return "Finished connectTcSE";
    }

    // Cannot directly call this method, itâ€™s private. Method to call must be
public.
    private void thisIsPrivateMethod(String[] someArgs) {
        System.out.println("TestRunJavaClass.thisIsPrivateMethod: cannot call this");
    }

    // Cannot directly call this method, itâ€™s private. Method to call must be
public.
    private static void printArgs(String[] someArgs) {
        printIt("printArgs");
        if (someArgs == null) {
            System.out.println(" > args is null");
        } else if (someArgs.length < 1) {
            System.out.println(" > args length is zero");
        } else {
            int idx = someArgs.length;
```

```
            for (int i=0; i<idx; i++) {
                System.out.println(" > arg[" + i + "] = " + someArgs[i]);
            }
        }
    }

    // Cannot call this method, wrong return type. Must return void, String or
String[].
    public int returnInt() {
        System.out.println("TestRunJavaClass.returnInt: cannot get here, wrong return
type");
        return 41;
    }

    // Cannot call this method, wrong number of parameters. Must have no parameters
    // or one String[] parameter.
    public void doSomething(String[] someArgs, String twoParams) {
        System.out.println("TestRunJavaClass.doSomething: cannot get here,
                           wrong number of parameters");
    }



    // Cannot call this method, wrong type of parameter. Must have no parameters or
    // one String[] parameter
    public void doSomething(int someArgs) {
        System.out.println("TestRunJavaClass.doSomething: cannot get here,
                          wrong type of parameter");
    }

    // Cannot call this method, itâ€™s private
    private static void printIt(String methodName) {
        String msg = "inside method TestRunJavaClass." + methodName;
        System.out.println(msg);
    }
    // Cannot call this method, itâ€™s private
    private static void showDialog(String text) {
        JOptionPane.showMessageDialog(null, text);
    }
}
```

Running an Activator From the Command Line With the tcradmin Script

You can enter the **tcradmin** script on the command line to run an Architect/Requirements activator.

The following example shows the **tcradmin** syntax:

```
tcradmin -action runActivator -script <ActivatorID> -selected <listOfParameters> -
user <username> -password <password> [-key <installationKey>]
```

Table 6-1 describes the **tcradmin** arguments.

Table 6-1. tcradmin Arguments for Running an Activator From the Command Line

Argument	Description
`-script`	LOID of the activator to run.
`–selected`	Comma-separated list of database object LOIDs or other information. This information is passed to the activator as the global variable **selected**.
`–user`	Architect/Requirements user name under which you want to log in to the server.
`-password`	Architect/Requirements password for the user name under which you want to log in to the server.
`–key`	Web server installation identifier used to look up Web application parameters such as **ImportExportDir**. This argument defaults to *machinename* + **"1"**, which is correct in most circumstances. This argument may be needed if there are no or more than one Architect/Requirements Web servers installed on this machine.

Using Macros

A macro is a type of activator that is run from a menu command rather than being triggered by an event. A macro specifies what input values, if any, it requires and the user interface will prompt for those values when the macro runs. The objects selected in the user interface are also passed into the macro.

Creating a Macro

See Access Privileges for an Activator for information about access privileges required to create, edit, or view an activator, which are also applicable to a macro.

1. Select the activator folder in the administration module and then choose **New** then **Activator** from the **File** menu.
2. Edit the activator's **Subtype** property by setting it to **Macro**.
3. If the macro requires user input, edit the activator's **Form Values** property, and select the values for which to prompt the user.
4. Open the activator and enter the Tcl script.

Form Values

The **Form Values** property on a macro allows you to specify the values to prompt the user for when the macro runs. Form Values is a multichoice property whose choices are the names of all the properties in the project. If the **Form Value** property is not empty, the user interface displays a dialog window to accept user input when the macro runs. Each property specified in **Form Values** has an entry field in the dialog window. The entry fields accept values based on the property type, text, choice, numeric, or date. The values entered by the user are passed into the Tcl environment using a global array name **tcrProperties**. If none of the existing properties is appropriate for the macro, you can create new properties.

Tcl Environment

Information from the user is passed into the Tcl interpreter as global variables. In addition to the normal variables passed into activators, there are two extra variables for macros:

- **selected:** List of objects selected in the user interface.
- **tcrProperties:** Array of values entered by the user. The array index is the property name. The value is what the user entered for the property. There is an entry for each property specified in the **Form Values**.

Event related variables that are passed into activators are not available in macros; these are the **currentObject**, event and **changeList**.

Running a Macro

Macros are run from the requirements module as follows:

For additional information, see the *Systems Architect/Requirements Management User's Manual*.

1. Choose **Run Macro** from the **Tools** menu.

   The command displays a list of all the macros in the project.

2. Select the desired macro and click **OK**.
3. If the macro requires user input a dialog box appears. Enter values for each property in the macros **Form Values**.

Macro Examples

This section provides examples of macros.

Setting a Property on Several Objects at the Same Time

Figure 6-6 demonstrates setting the **AssignedTo** property on several objects at once. The example assumes an **AssignedTo** property exists in the project and that the macro's **Form Values** is set to **AssignedTo**.

```
# get the value the user entered for AssignedTo
set assigned $tcrProperties(AssignedTo)

# iterate over the objects selected in the user interface
foreach object $selected {
        # set the AssignedTo property to the user entered value
        setValue $object AssignedTo $assigned
}
```

Figure 6-6. Setting a Property on Several Objects at the Same Time

Setting Several Properties on Multiple Objects

Figure 6-7 demonstrates setting any number of user entered properties on the selected objects. The properties are specified in the macro's **Form Values**.

```
# iterate over the objects selected in the user interface
foreach object $selected {
    foreach property [array names tcrProperties] {
        # set the assignedTo property to the user entered value
        setValue $object $property $tcrProperties($property)
    }
}
```

Figure 6-7. Setting Several Properties on Multiple Objects

Creating a Derived Requirement

Figure 6-8 demonstrates how to create a derived requirement for the selected source requirement. The derived requirement has a trace link to the source and some property values are set automatically. The example assumes a **Derived Requirement** subtype exists in the project.

```
set sourceReq $selected ; # assumes 1 requirement is selected

       # create a derived requirement owned by the selected requirement
       set derived [createObject “” $sourceReq “Derived Requirement”]

       # copy property values from the source to the derived requirement
       setValue $derived Name [getValue $sourceReq Name]
       setValue $derived MHTML [getValue $sourceReq MHTML_FULL]  ; # sets the
body text

       # set the assignedTo property to the current user
       setValue $derived assignedTo [getValue $tcrUser Name]

       # create a trace link to the derived requirement
       createLinks $sourceReq $derived “Complying”
```

Figure 6-8. Creating a Derived Requirement

Collecting Metrics on Requirements

Figure 6-9 iterates over requirements, beneath the selected object, and gather statistics. The example assumes a status property is assigned to requirements in this project.

```
# get the requirements in the selected project or folder
       set reqs [search $selected "*" "" {RequirementDB}]

# iterate over the requirements and check their status
       set complete 0
       set total [llength $reqs]
foreach req $reqs {
       if {[getValue $req Status] == "Completed"} {
         incr complete
   }
}

displayMessage "Status of requirements in [getValue $selected Name]"
displayMessage "Total requirements: $total"
displayMessage "Completed requirements: $complete"
displayMessage "Percent complete: [expr $complete * 100.0 / $total]"
```

Figure 6-9. Collecting Metrics on Requirements

Working with Shortcut Objects

Shortcuts act as a separate occurrence of an object in a different location. When processing objects using the Systems Architect/Requirements Management API methods, you may encounter shortcuts. For example, the `getList MEMBER_LIST` call on a folder or other parent object returns shortcut members along with the rest of the true objects.

Shortcuts act in many ways like the actual object. The API forwards most **getValue**, **setValue**, and **getList** methods to the actual object. In some cases, it may be necessary to determine that you are working with shortcuts and handle them specially.

For this special handling, shortcuts support several properties, such as:

- **Shortcut** indicates whether this object is a shortcut ("true") or is not ("").
- **Original Object** is the LOID of the actual object.
- **Shortcut Options** is a property with flags to choose the behavior of shortcuts, such as whether to allow the shortcut to be expanded to show the actual objects children.

The following code recognizes that an object is a shortcut and gets the actual object:

```
if {[getValue $obj Shortcut] == "true"} {
    set actualObj [getValue $obj "Original Object"] }
```

There are some properties that are retrieved directly from the shortcut, instead of being forwarded to the actual object in the **getValue** and **setValue** methods. A shortcut's local properties are:

- **Owner** is the LOID of the shortcut's owner.
- **Owner Name** is the name of the shortcut's owner.
- **Security Profile** is the profile on the shortcut itself, if any.
- **WhereUsed Object Count** is the number of objects that reference the object. For shortcuts, this count is typically zero.
- **Used Object Count** is the number of objects that the object references. For shortcuts, this count is always one.
- **Cross Project Link** is true if the shortcut references an object in a different project and is false if the referenced object is in the same project.
- **Project** is the name of the project that owns the shortcut.
- **Full Name** is the full name of the shortcut, including the object that owns it.
- **Number** is the hierarchy number of the shortcut. For example, **2.3.7**.
- **Local Number** is the last portion of the hierarchy number of the shortcut. For example, **7**.

Some **getList** keywords are processed directly by the shortcut. All other **getList** keywords are retrieved from the original object.

- **MEMBER_LIST** is empty if the shortcut does not inherit children. Otherwise, the **getList** method is forwarded to the primary object to retrieve its members.

A shortcut never has children of its own. It can (optionally) inherit the children of the primary object that it references. In that case, there are no extra shortcut objects for those children. The actual children of the primary object are displayed below the shortcut in

the Systems Architect/Requirements Management client. The small red-arrow icon indicates they were reached via a shortcut. If a shortcut inherits the primary object's children, the **getList** method with **MEMBER_LIST** keyword on a shortcut returns the exact same objects as the **getList** method on the primary object.

- **WHERE_USED_LINK_LIST** is a list of where-used links to the object. This list is typically empty for shortcuts.
- **WHERE_USED_OBJECT_LIST** is a list of objects from following the where-used links from the object. This list is typically empty for shortcuts.
- **USES_LINK_LIST** is a list of reference links from the object. This list always has one entry for shortcuts, that is the link to its primary object.
- **USES_OBJECT_LIST** is a list of objects referenced by the object. This list always has one entry for shortcuts, that references its primary object.

Activators are not triggered on shortcut objects because there is no type definition for the shortcut type. However, many actions on a shortcut are applied to the original object. These actions trigger events on the original object, not the shortcut.

Shortcuts cannot handle the Create, Before Delete, and After Delete events because shortcuts cannot have their own activators. However, when shortcuts are created and deleted, there are some observable events on other objects. They are:

- When a shortcut is created, the After Relation (Add Member) event on the shortcut's owner and the After Relation (Add Where Used) event on the object the shortcut references.
- When a shortcut is deleted, the After Relation (Remove Member) event on the shortcut's owner.

There is no event on the referenced object when a shortcut is deleted. There is also no event on the shortcut's owner if it is restored from the recycle bin.

Working with Reference Links

When a reference is inserted into the text of a requirement or note using the **Copy Reference Link** command, a symbolic reference link object is created. This section describes how to access and manipulate these references.

The **getList** keywords related to references are (see `DataBean.LIST`):

- **WHERE_REFERENCED_LIST** is the list of the requirements and notes that have references to the input (this) object.
- **WHERE_REFERENCED_LINKS_LIST** is the list of symbolic links that reference the input (this) object.
- **REFERENCING_LIST** is the list of objects the input (this) note or requirement references.
- **REFERENCING_LINKS_LIST** is the list of symbolic reference links from the input (this) note or requirement.

The **getValue** keywords related to symbolic link objects are (see `DataBean.PROPERTY`):

- **START_OBJECT** is the requirement or note containing the reference.
- **END_OBJECT** is the referenced object.
- **PROPERTY** is the name of the referenced property.

References to Versioned Objects

When a reference is made to a versioned object, Systems Architect/Requirements Management decides which version of the object to use. There are two modes of behavior. If the **Promote References** option is selected in the project's **Project Settings** property, then references use the effective version. Otherwise, references use a specific version. In either case, there is one symbolic link object that points to a specific version. When **Promote References** is enabled, the reference is shifted to the effective version dynamically.

The **getValue** and **setValue** keywords that apply to symbolic links when versions are involved are:

- **USED_OBJECTS**: Gets the referenced object (like **END_OBJECT**). **END_OBJECT** returns the specifically referenced version. **USED_OBJECTS** returns the effective version of the referenced object if **Promote References** is enabled.

  USED_OBJECTS is also supported in **setValue** which allows adjusting which version is specifically referenced.

- **REFERENCE_SETTINGS**: By default, reference links use the behavior specified in the **Project Settings** property. The default behavior can be overridden for a specific symbolic link by setting the **REFERENCE_SETTINGS** property with **Promote References** or **Fixed References** as value. A blank value indicates the default project behavior.

The **getList** keywords that apply when versions are involved are:

- **WHERE_REFERENCED_REAL**: Returns the requirements and notes that reference the input (this) object (like **WHERE_REFERENCED_LIST**). **WHERE_REFERENCED_LIST** takes the **Promote References** option into account where as **WHERE_REFERENCED_REAL** returns specific references.
- **WHERE_REFERENCED_LINKS_REAL**: Returns symbolic links that reference the input (this) object (like **WHERE_REFERENCED_LINKS_LIST**).

WHERE_REFERENCED_LINKS_LIST takes the **Promote References** option into account where as **WHERE_REFERENCED_LINKS_REAL** returns specific references.

- **REFERENCING_REAL**: Returns the objects the input (this) note or requirement references (like **REFERENCING_LIST**). **REFERENCING_LIST** takes the **Promote references** option into account where as **REFERENCING_REAL** only returns specific references.

Chapter 7: Using Icon Overlay

Icon overlay functionality provides a mechanism to add custom overlays to icons. You can use the functionality to overlay an existing icon or completely change an icon.

Container objects such as folders, building blocks, requirements, and projects have a new **hidden** property.

If there is a value on this property, and if it is the LOID of a type definition, then the icon that is set for that type definition will be used as an overlay to the current icon. The custom overlay is the first overlay, followed by any possible out of the box overlays such as subtypes, variants, or frozen.

A new **icon** type definition has been added. You can create subtypes of the **icon** type definition for your icons.

Following is a sample code for Icon Overlay using Tcl:

```
# get the LOID of the icon overlay
set name "BigRedEx"
# find type def by iterating over all type defs
 foreach def [getList $currentProject OBJECT_TYPE_LIST] {
       if {[getValue $def Name] == $name} {
              set iconOverlay $def
              break
       }
}
# the code must ensure that icon was found before continuing
 setValue $selected "Icon Overlay" $iconOverlay
```

You can create activators (after modifying) that can set the **Icon Overlay** property based on the criteria that you determine.

For example:

- If it is assigned to Jerry and it is complete then make it a big green **J**.
- If it is overdue, then mark it with a big red exclamation mark.

A sample icon overlay is given in Figure 7-1.

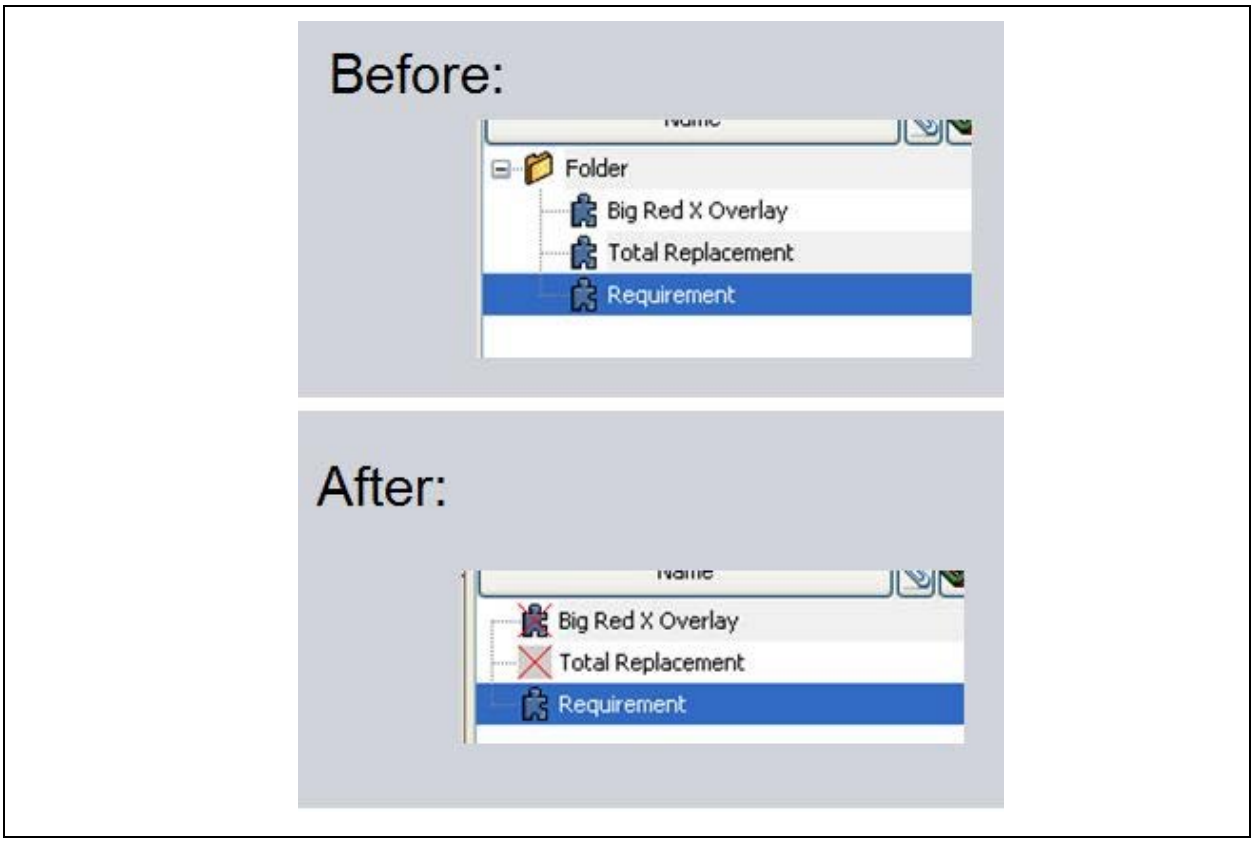


Figure 7-1. Icon Overlay sample

In Figure 7-1, the first requirement has a red **X** overlayed on the requirement. The second requirement is completely replaced with a red **X**.

Any overlay change is always a change on the actual object. The overlay is saved in the database and shared with all users in the Architect/Requirements project.

Chapter 8: Using Generic Links

This chapter describes generic links, and explains how to use them.

Introduction

The Systems Architect/Requirements Management links have built-in behavior that cannot be customized for new applications. New interfaces that require novel link behavior need a link type whose behavior can be completely controlled. Generic links provide the ability to control and customize the behavior of links between the Systems Architect/Requirements Management objects.

Generic link is a new database object type that can be subtyped to support application-specific linking needs. The default behavior and supported operations are similar to that of the trace links. For more information about the generic links, see the *Systems Architect/Requirements Management User's Manual*.

Support for Generic Links

Generic links are supported by attributes similar to those of the trace links. Table 8-1 lists the attributes supported for generic links and their equivalent trace link attributes. For more information about the trace links and their attributes, see the *Systems Architect/Requirements Management User's Manual*.

Table 8-1. Attributes Supported for Generic Links

Generic Links	Trace Links
Incoming Objects	**Complying Objects**
Incoming Link List	**Complying Link List**
Incoming Object Count	**Complying Objects Count**
Generic Link Count	**Trace Link Count**
Outgoing Objects	**Defining Objects**
Outgoing Link List	**Defining Link List**
Outgoing Objects Count	**Defining Objects Count**

Examples of API Methods

You can use the application-specific API methods to create, delete, modify, and navigate generic links. Generic links are supported in the createLinks, deleteLinks, setValue, getValue, and getList API methods.

- To create a generic link between two objects:

  ```
  createLinks $start $end "Outgoing Generic Link"
  ```

  You can also use `"Incoming Generic Link"`, however, the direction of the link is reversed.

- To create a subtype of a generic link:

  ```
  createLinks $start $end "Outgoing Generic Link" "mysubtype"
  ```

- To get the list of incoming objects:

  ```
  getList[$obj "INCOMING_OBJECT_LIST"]
  ```

  This command is similar to the `getList[$obj "COMPLYING_OBJECT_LIST"]` trace link command.

- To get the outgoing link list:

  ```
  getList[$obj "OUTGOING_LINK_LIST"]
  ```

  This command is similar to the `getList[$obj "DEFINING_LINK_LIST"]` trace link command.

When used with the getList API method:

- `LINK_TO_LIST` and `LINK_FROM_LIST` get the `INCOMING_OBJECT_LIST` and `OUTGOING_OBJECT_LIST` objects, respectively.
- `TO_LINKS` and `FROM_LINKS` get the `INCOMING_LINK_LIST` and `OUTGOING_LINK_LIST` objects, respectively.

Chapter 9: Cross-Product Messaging

This chapter describes the new `proxyAction` service which has been added to the set of SOAP services implemented for proxy objects in both Systems Architect/Requirements Management and Teamcenter Engineering. It uses the same URL, SOAP message format, authentication, session management and infrastructure as the existing set of services.

proxyAction

Each `proxyAction` request sends a single string argument to a foreign application. Based on the URL found in Application Registry for an Application ID, the string argument is sent to the foreign application. The response is a single string result, and a success/error indicator.

The applications must not place any restrictions on the content of the message or response strings, but they must be encoded properly to pass across the SOAP transport.

The `proxyAction` request message is of this form:

`operationID` – proxyAction

`inParam1`, “key” STRING – authentication key

`inParam2`, “appGuid” STRING – the sending application’s GUID

`inParam3`, “inputMsg” STRING – the action input string

The `proxyAction` response is of this form:

`outParam1`, “errMsg”, string – expected value Success

`outParam2`, “outputMsg”, string – the formatted output string

Systems Architect/Requirements Management and Teamcenter Engineering must each be able to send and receive `proxyAction` requests.

Setting Up proxyAction in Systems Architect/Requirements Management

Incoming proxyAction Requests

Incoming `proxyAction` requests are handled by first performing the usual authentication steps, and then connecting to an entry from the Systems Architect/Requirements Management session pool, as with any incoming Proxy SOAP request.

The incoming `proxyAction` string is sent to the Tcl interpreter to be evaluated. That Tcl may ultimately call any of the Tcl API functions, although it most often simply runs an existing Activator. The Tcl string result is returned to the originator of the proxyAction request, along with a success indication. If the Tcl interpreter exits with an unhandled error, the Tcl error string is returned as the string result, along with an error indication.

Sending proxyAction Requests

Sending `proxyAction` requests must be supported through a new Tcl API call:

```
proxyAction appID message
```

Where:

`appID` – A valid WOLF Application ID, likely found by reading the value from a Remote Proxy object within Systems Architect/Requirements Management.

`message` – The string to be sent to the remote application.

If the `appID` is not registered with WOLF, or if the remote application returns an error, a Tcl error should be thrown, with the Tcl error string set with the result from the remote application. If the call succeeds, the result string from this procedure should be the response string returned from the remote application.

Setting Up proxyAction in Teamcenter Engineering

Refer to Teamcenter Engineering documentation for details.

Remote Proxy Objects and Tcl API

Get Remote Proxy Properties

Tcl API callers must be able to get the Remote Proxy list of **ApplicationIDs** and **ObjectIDs** for any Systems Architect/Requirements Management object.

```
getValue object REMOTE_PROXY
```

The result is a list of lists of strings. The outer list has an entry for each Remote Proxy for the target object, or an empty list if it has none. Each inner list has two string elements, an **ApplicationID** and an **ObjectID**.

Appendix A: Word Content Activators

The text content of requirements and notes is processed using Microsoft Word's single file web page (**mhtml**) format. A fragment of the **mhtml** file, containing only the content of the requirement or note, is stored in the database with each object. When Word export is run, **mhtml** fragments from the exported objects are concatenated together, along with other data such as the style sheet, to form the **mhtml** file. An **mhtml** fragment can have references to objects outside the fragment. For example, cross references, which point to information in another Architect/Requirements object, and style names, which reference a style definition in the stylesheet.

As the export file is built, some of these references may require adjustments to function correctly. For example, the internal names used for graphic references. The graphic names include a sequential number to keep them unique. For example, **image001.gif**. The same graphic name can be used for different graphics in different Architect/Requirements objects. The graphic names are modified during export to ensure uniqueness.

In certain cases, it is not possible to resolve the external references. For example, a cross reference is not functional if the target for the reference is not included in the export. A custom style name does not work if that style is not defined in the style sheet used for the export.

Architect/Requirements does not correct external reference for numbered and bulleted lists. This is due to the unusual way in which numbered and bulleted list styles are defined in MS Word **mhtml** format. List types are arbitrarily assigned a list number and the mapping from list number to list style is not always clear. This causes the wrong list type to be used with numbered and bulleted lists in exports.

As a work around, activator events are defined to enable customized MS Word content correction. Activators can fire both during and after **mhtml** file generation. For specific use cases it is possible to correct list styles or MS Word content issues.

The new activator events are:

- **Word Edit Pre-Processor**
- **Word Edit Post-Processor**
- **Word Export Pre-Processor**
- **Word Export Post-Processor**

Unlike the other activators, these activators trigger based on the names. If an activator with the event name exists in a project, it triggers at the appropriate time during **mhtml** file generation for exports and edits in that project.

Word Edit Pre-Processor

This activator triggers when a requirement or note is opened for edit in MS Word. This allows modifying the **mhtml** file content that is displayed in MS Word.

The following information passed to the activator:

currentObject – Object ID of the requirement or note being opened for edit.

selected[0] – the **mhtml** file content.

The activator return value is used as the object's **mhtml** file content. If an error occurs in the activator or the return value is blank then the activator result is ignored.

Word Edit Post-Processor

This activator triggers after a requirement or note is saved in MS Word and before the content is saved in the database. This allows modifying the HTML content that gets saved in the database.

The following information passed to the activator:

currentObject – Object ID of the requirement or note being saved.

selected[0] – the HTML content being saved.

selected[1] – the full **mhtml** file that was edited.

The activators return value is stored as the object's HTML content. If an error occurs in the activator or the return value is blank then the activator result is ignored.

The **mhtml** file is provided so that you can examine the stylesheet information.

Word Export Pre-Processor

This activator triggers for each database object before it is added to the exported MS Word document. This allow modifying object content in the document.

The following information passed to the activator:

currentObject – ID of the object being exported.

selected[0] – HTML content of the object.

selected[1] – ID of the style sheet used for the export.

The returned value is added to the export document instead of the original content. If an error occurs in the activator or the return value is blank then the activator result is ignored.

The HTML content passed has already been processed using the object template and includes the heading.

This activator can trigger for all objects types and not for requirements and notes only.

Word Export Post-Processor

This activator triggers once for an exported MS Word document. This allows modifying document file content before it is downloaded to the client and opened.

The following information passed to the activator:

currentObject – ID of the object being exported. If multiple objects are exported then the first object is used.

selected[0] – Path name of the exported MS Word document.

The exported file name is provided so that you can examine and modify the **mhtml** file.

The activator return value is not used. Errors in the activator are logged and ignored.

Appendix B: Examples for using C# API's

This chapter contains examples of using the Architect/Requirements API's. You can use the API using VBA and C#.

Accessing Using VBA

An example of using API's to access Architect/Requirements using VBA is provided with the kit. The example uses API's to perform the following functions:

- Connection to Architect/Requirements
- Get a list of projects
- Create folder in the selected project
- Create building block in a folder
- Modify an object
- Delete an object from Architect/Requirements
- Searching for requirements

To view the example:

1. Extract the *<DVD Root>*\ **ClientAPIExample\ClientAPIExample.zip** file from the product DVD to a temporary folder on the machine where you are running the client.
2. Double click the **ClientAPIExample.xlsm** to open it in Microsoft Excel.
3. If you get a security warning, click **Enable Content** to enable the macros in the project.
4. Click the **Launch Example** button.
5. Enter the port number on which the client can be accessed and click **OK**.

   You can also accept the default port **4002**.
6. Enter the name of the machine on which the Architect/Requirements server is running and click **OK**.
7. Enter the Architect/Requirements user name to access the application and click **OK**.
8. Enter the password for the account and click **OK**.

   If you have entered the correct logon credentials, a connection successful message is displayed.

9. Click **OK**.
10. The **Select Project** dialog box displays a list of projects in Architect/Requirements. From the drop down list, select the project in which you want to create the objects and click **OK**.
11. Various Architect/Requirements objects are created and corresponding success or failure messages are displayed.
12. Architect/Requirements is searched and the requirements are displayed.
13. The example launches automatically and connects the Architect/Requirements application, creates a project, creates a requirement.

To view the VBA code, enable the **Developer** ribbon.

1. Click **File→Options→Customize Ribbon**, select the **Developer** check box, and click **OK**.
2. Click the **Developer** ribbon and click Visual Basic.
3. Double-click **ThisWorkbook** under the Microsoft Excel Objects in the left pane.
4. To close the example, exit Excel.

Following functions are defined in the example to perform various operations:

You must provide the parameters corresponding to your setup for the functions to work correctly.

- **connectToTcR()**

  Function to connect to Architect/Requirements. This function runs once when the example is launched. The following parameters are collected from the user to connect to Architect/Requirements:

  - Port on which the client/server is running

    The **connectionService.tcr_client_socket_port = socketNum** paramter is used to set the port number.

  - Name of the server machine

    The **connectionService.tcr_client_socket_url = machineNum** paramter is used to set the name of the Architect/Requirements server.

  - Valid Architect/Requirements username and password

    The **connectionService.user_name = uName** paramter is used to set the username of the Architect/Requirements user.

    The **connectionService.user_password = pass** paramter is used to set the password for the user.

  The function attempts to connect to the Architect/Requirements client. If the client is not running, it connects to the server.

- **getProjects()**

  Function to get a list of projects from Architect/Requirements.

- **getSelectedProjectId(projectName As String)**

  Function to get the ID of the selected project. This function accepts the project name and gets the project ID.

- **createObjectInTcR(owner As String, typeName As String)**

  Function to create any Architect/Requirements object. This function accepts the ID of the object that you want to create and the type of the object.

- **modifyObjectInTcR(modiftObject() As TcR.dataBean)**

  Function to modify objects in Architect/Requirements. This function accepts the object to be modified. It modifies the name and text of the requirement, however, this function can be used to modify any object created in Architect/Requirements.

- **deleteObjectInTcR(objectId() As String)**

  Function to delete objects in Architect/Requirements. This function accepts the ID of the object that you want to delete.

Accessing Using C#

An example of using API's to access Architect/Requirements using C# is provided with the kit. The example uses API's to perform the following functions:

- Connection to Architect/Requirements
- Get a list of projects
- Create folder in the selected project
- Create building block in a folder
- Modify an object
- Delete an object from Architect/Requirements
- Searching for requirements

You must have Microsoft Visual Studio 2010 installed to view the example.

To view the example:

1. Extract the *DVD Root***ClientAPIExample\ClientAPIExample.zip** file from the product DVD to a temporary folder on the machine where you are running the client.
2. Navigate to **Example in c#** folder.
3. Double click **API Reference Solution.sln**.
4. In the **Solution Explorer**, click the **MainClass.cs**.
5. Modify the following paramters to match your installation:
   - **connectionService.tcr_client_socket_port**
   - **connectionService.tcr_client_socket_url**
   - **connectionService.tcr_server_controller_url**
   - **connectionService.user_name**
   - **connectionService.user_password**
6. Click the **Start** button on the toolbar to view the functions of the example.
7. To close the example, exit Visual Studio.

API Reference Index