Optimal Trace

Plug-in SDK Reference

Please direct questions about Optimal Trace or comments on this document to:

Technology Customer Support
Compuware Corporation
Customer Support Hotline
1-800-538-7822
FrontLine Support Web Site:
http://frontline.compuware.com

For telephone numbers in other geographies, see the list of worldwide offices at http://www.compuware.com.

Local Build: April 9, 2009, 8:23

Contents

CHAPTER 1

Using the Optimal Trace Plug-in SDK

With the Optimal Trace Plug-in SDK, you can create custom plug-ins that integrate Optimal Trace with external applications or serve as functions that automate repetitive actions for accessing and manipulating Optimal Trace requirements data. Plug-ins are Java components that are accessible from within the Optimal Trace client. Development of Optimal Trace plug-ins requires Java programming skills.

The Optimal Trace Plug-in SDK enables create, modify, and delete capability for the following Optimal Trace entities:

- Project custom properties
- Glossary entries
- Actors
- Packages and their custom properties
- Structured requirements and their custom properties
- Use case steps and their custom properties including main and alternate scenarios
- Refinements
- Links
- Simple requirements and their custom properties
- Items and their custom properties
- Notes
- Branches

NOTE

Creating an Optimal Trace project or deleting an existing Optimal Trace project is not supported. You can, however, update the project name and description.

Installation Location

When you install Optimal Trace Enterprise, the Optimal Trace Plug-in API (`OTAPI.JAR`) is automatically installed in the following Optimal Trace Enterprise installation folder:

```
<Install_DIR>\Program Files\Compuware\Optimal Trace\Optimal Trace Enterprise Edition\Lib\OTAPI.JAR
```

CHAPTER 2

Creating and Deploying an Optimal Trace Plug-in

This topic describes the process for creating and deploying an Optimal Trace plug-in.

1. In the following location of the Optimal Trace installation folder, create a folder in which to save the required Optimal Trace plug-in files :

   `<USER_DIR>\Program Files\Compuware\Optimal Trace\Optimal Trace Enterprise Edition\plugins\<PLUGIN_NAME>`

   NOTE

   This is the default location for Optimal Trace plug-ins. To change the default location of the `plugins` folder, you must change the value of the `pluginsDir` property in the `default.properties` file. This file appears in the following location of the Optimal Trace installation folder:

   `<USER_DIR>\Program Files\Compuware\Optimal Trace\Optimal Trace Enterprise Edition\plugins\`.

2. Create and compile a JAR file containing the required Java class files of the plug-in. Save the JAR file in the `<PLUGIN_NAME>` folder.
3. Create a `plugins.properties` file to store information about the plug-in. Save this file in the `<PLUGIN_NAME>` folder.
   For example:

   ```
   pluginName=myCustomOTPlugin
   pluginVersion=1.0.0
   ```

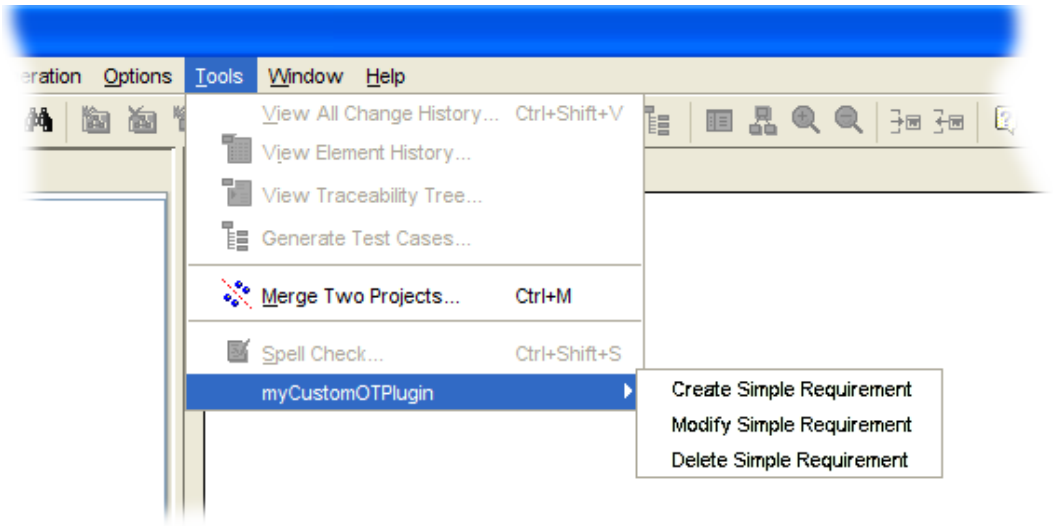
4. Create an `actions.xml` file to add menu commands for the plug-in to the **Tools** menu of the Optimal Trace menu bar. Save this file in the `<PLUGIN_NAME>` folder.

   The following example shows how to add menu commands to access custom actions defined in the plug-in `myCustomOTPlugin`:

   ```
   <ActionGroups>
     <PluginActionGroup>
       <name>myCustomOTPlugin</name>
       <Action>CreateSimpleReqAction</Action>
       <Action>ModifySimpleReqAction</Action>
   ```

```
    <Action>DeleteSimpleReqAction</Action>
  </PluginActionGroup>
  <Actions>
    <CreateSimpleRequirementAction>
       <Action>com.myCustomOTPlugin.ot.plugins.CreateSimpleReqAction</Action>
       <Label>Create Simple Requirement</Label>
    </CreateSimpleRequirementAction>
    <ModifySimpleRequirementAction>
       <Action>com.myCustomOTPlugin.ot.plugins.ModifySimpleReqAction</Action>
       <Label>Modify Simple Requirement</Label>
    </ModifySimpleRequirementAction>
    <DeleteSimpleRequirementAction>
       <Action>com.myCustomOTPlugin.ot.plugins.DeleteSimpleReqAction</Action>
       <Label>Delete Simple Requirement</Label>
    </DeleteSimpleRequirementAction>
  </Actions>
</ActionGroups>
```

The resulting menu commands appear on the **Tools** menu as follows:



5. Start Optimal Trace.
   Optimal Trace scans the plugins folder, detects any plugins, and adds any menu commands specified in the `actions.xml` file to the **Tools** menu.

To use the menu commands, you must have the appropriate project selected in the tree view.

CHAPTER 3

About the Optimal Trace Plug-in API

This section provides instructions and examples for using the Optimal Trace Plug-in API to create, modify, and delete Optimal Trace entities. Detailed information about specific Java classes and interfaces supported in the Optimal Trace Plug-in API is available in JavaDoc format in the following folder external from this document:

`OTAPIJavaDoc`

Open the `OTAPIJavaDoc` folder and click `index.html` to load the Optimal Trace Plug-in API JavaDoc in a browser.

The Optimal Trace Plug-in API reference documentation is organized by entity type. Each entity type folder contains instructions and examples specific to the entity type. The provided instructions and examples for each entity type may not include instructions for all commonly performed actions. For example, you can create, modify, and delete custom properties for any entity type, however, instructions for doing so are not provided for every entity type as the process is the same for each entity type.

Projects

With the Optimal Trace Plug-in API, you can modify the name and description of an existing project and save a local or remote project. You can also create, modify, and delete custom properties of a project.

This section provides specific instructions for modifying the name of a project and its description. Additionally, this section contains instructions for creating, modifying, and deleting custom properties of a project.

NOTE

Creating an Optimal Trace project or deleting an existing Optimal Trace project is not supported. You can, however, update the project name and description.

Modifying a Project Name and Description

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Lock the project object to maintain integrity for simultaneous updates.

```
//Pass the project and the object that needs to be locked
LockSupport.lock(pifc, pifc)
```

3. Modify the name and description as desired.

```
//Call the setName and pass the new name as a string parameter
pifc.setName("New Name");

//Call setLongDescription and pass the new description as a string parameter
pifc.setLongDescription("New Description");
```

4. Save the project. For more information, see Saving a Project [p. 12].

Creating a Custom Property for a Project

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Get the project's `CustomPropertyTemplateList` object.

```
//Get the instance of the object from the current project
CustomPropertyTemplateListIfc attribList = pifc.getCustomPropertyTemplates();
```

3. Place a lock on the `CustomPropertyTemplateList` object of type list and holders.

```
//Use the lock constants to place a lock of type of list and holders
LockSupport.lock(pifc,attribList ,
LockSupport.CUSTOM_PROPERTY_TEMPLATE_LIST_AND_HOLDERS);
```

4. Create a new custom property and assign its name and definition.

```
//Use the custom property support class to set the name and definition
CustomPropertyTemplateIfc cptemplate =
CustomPropertySupport.getInstance().createCustomPropertyTemplate(attribList);
cptemplate.setName("New Custom Property");
cptemplate.setDefinition("New Custom Property Value");
```

Modifying a Custom Property of a Project

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Get the project's `CustomPropertyTemplateList` object

```
//Get the instance of the object from the current project
CustomPropertyTemplateListIfc attribList = pifc.getCustomPropertyTemplates();
```

3. Get the project's custom properties by iterating and placing a lock before modifying the value of the desired custom property.

```
//Get the list of CP's from the current project.
CustomPropertyTemplateListIfc attribList = pifc.getCustomPropertyTemplates();
for (Iterator iterator = attribList.getCustomPropertyTemplates().iterator() ;
iterator.hasNext();)
{
   CustomPropertyTemplateIfc cptemplate = (CustomPropertyTemplateIfc) iterator.next();

   //Find the match
   if(cptemplate.getName().equalsIgnoreCase("New Custom Property"))
   {
      //Place the lock of type list and holders.
      LockSupport.lock(pifc,attribList ,
LockSupport.CUSTOM_PROPERTY_TEMPLATE_LIST_AND_HOLDERS);

      //Set Definition of the CP to new value
      cptemplate.setDefinition("new updated value");
      return;
   }
}
```

4. Save the project. For more information, see Saving a Project [p. 12].

Deleting a Custom Property of a Project

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Get the project's `CustomPropertyTemplateList` object.

```
//Get the instance of the object from the current project
CustomPropertyTemplateListIfc attribList = pifc.getCustomPropertyTemplates();
```

3. Get the required custom property by iterating and placing a lock on it. Next, delete the custom property.

```
//Get the list of custom properties from the current project.
CustomPropertyTemplateListIfc attribList = pifc.getCustomPropertyTemplates();
for (Iterator iterator = attribList.getCustomPropertyTemplates().iterator();
iterator.hasNext();)
{
    CustomPropertyTemplateIfc cptemplate =
    (CustomPropertyTemplateIfc) iterator.next();

    //Find the match
    if(cptemplate.getName().equalsIgnoreCase("My Custom Property"))
    {
        //Place the lock of type list and holders.
        LockSupport.lock(pifc,attribList,
        LockSupport.CUSTOM_PROPERTY_TEMPLATE_LIST_AND_HOLDERS);

        //Delete using the support class
        CustomPropertySupport.getInstance().
        deleteCustomPropertyTemplate(cptemplate, attribList);
        return;
    }
}
```

Saving a Project

The method for saving a project using the Optimal Trace Plug-in API is different for local projects and remote projects. The following steps described how to save each type of project using the Optimal Trace Plug-in API.

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Save the project.
   - For local projects:

```
//Get the instance of ProjectManagerFacade object, which is a singleton
ProjectManagerFacade pmf = ProjectManagerFacade.getInstance();

//To save a local project, get the location of the project first
String loc = pifc.getLocation();

//Using the ProjectManagerFacade, save the project by passing the location
//and the project object
pmf.saveProjectXML(pifc,loc);
```

   - For remote projects:

```
//Get the instance of ProjectmanagerFacade object, which is a singleton
ProjectManagerFacade pmf = ProjectManagerFacade.getInstance();

//Save the project using the ProjectManagerFacade
 pmf.saveProject(pifc);
```

Glossary Entries

With the Optimal Trace Plug-in API, you can create, modify, and delete glossary entries. You cannot, however, create or delete an entire glossary entity. Additionally, you can create, modify, and delete custom properties of glossary entries.

This section provides specific instructions for creating and deleting glossary entries.

Creating a Glossary Entry

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Get an instance of a `DictionaryDefIfc interface` object.

```
//Using DictionaryDefFactory, get an instance of an object.
DictionaryDefFactory  dicFac  =  DictionaryDefFactory.getInstance();
```

3. Get the selected project and glossary entity.

```
//Get the instance of a project container, which is a singleton
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//Get the selected project from the project container object
```

```
ProjectIfc   pifc = projCnt.getSelectedProject();
GlossaryIfc glossary = pifc.getGlossary();
```

4. Lock the glossary object.

```
//Pass the project and the object to be locked
LockSupport.lock(pifc, glossary);
```

5. Create a dictionary object and assign it to the glossary entity.

```
//Create a new DictionaryDefIfc object and assign the existing Glossary obj
//from the project to it
DictionaryDefIfc  newDef  =  dicFac.createNewDictionaryDef(glossary);

//Assign the glossary term name and definition
newDef.setName(“New Name”);
newDef.setDefinition(“New Definition”);
```

6. Save the project. For more information, see Saving a Project [p. 12].

Deleting a Glossary Entry

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Get the project in which the desired glossary entity appears.

```
//First get the instance of a project container, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From project container object get currently selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
GlossaryIfc gls = pifc.getGlossary();

//Get the list of DictionaryDefs from Glossary entity
java.util.List lst = gls.getDictionaryDefs();
```

3. Place a lock on the `ActorListIfc` object.

```
//Iterate through the list, find the match, and remove the entry
for (Iterator iterator = lst.iterator(); iterator.hasNext();)
{
   DictionaryDefIfc dictionaryDef1 = (DictionaryDefIfc) iterator.next();
   if(dictionaryDef1.getName().equals("My Glossary"))
   {
      //Pass the project and the objects to be locked
      LockSupport.lock(pifc, gls);
      LockSupport.lock(pifc, dictionaryDef1);

      //Use the DeleteSupport object to remove the custom properties for
      //the DictionaryDef entity; Pass the Glossary object and the definition
       DeleteSupport.getInstance().deleteDictionaryDef(gls,dictionaryDef1);
       return;
   }
}
```

4. Get the `ActorListIfc` object.

```
//First get the instance of a project container, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From project container object get currently selected project and then ActorList
ProjectIfc   pifc = projCnt.getSelectedProject();
ActorListIfc alifc = pifc.getActorList ();
```

5. Save the project. For more information, see Saving a Project [p. 12].

Actors

With the Optimal Trace Plug-in API, you can create, modify, and delete actors. Additionally, you can create, modify, and delete custom properties of a actor.

This section provides specific instructions for creating and deleting actors.

Creating an Actor

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Get the `ActorListIfc` object from the selected project.

```
//First get the instance of a project container, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//Get the selected project and ActorList from the project container object
ProjectIfc   pifc = projCnt.getSelectedProject();
ActorListIfc alifc = pifc.getActorList ();
```

3. Place a lock on the `ActorListIfc` object.

```
//Pass the project and the object to be locked
LockSupport.lock(pifc, alifc);
```

4. Create a new `ActorIfc` object and pass the `ActorListIfc` object as a parameter.

```
//Set the actor name and definition
ActorIfc newActor = ActorFactory.getInstance().createNewActor(alifc);
newActor.setName("New Actor Name");
newActor.setDefinition(“New Actor Name Value”);
```

5. Save the project. For more information, see Saving a Project [p. 12].

Deleting an Actor

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Get the `ActorListIfc` object.

```
//Get the instance of a project container, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//Get selected project and ActorList from the project container object
ProjectIfc   pifc = projCnt.getSelectedProject();
ActorListIfc alifc = pifc.getActorList ();
```

3. Place a lock on the `ActorListIfc` object.

```
//Pass the project and the object to be locked
LockSupport.lock(pifc, alifc);
```

4. From the `ActorListIfc` object, delete the desired actor.

```
//Find the Actor to delete and use the DeleteSupport object
for (Iterator iterator = alifc.getActors().iterator() ; iterator.hasNext();)
```

```
{
   ActorIfc actor = (ActorIfc) iterator.next();
   if(actor.getName().equalsIgnoreCase("My Actor"))
   {
      //Now place lock of type DELETE_ACTOR_LOCK
      LockSupport.lock(pifc,actor,LockSupport.DELETE_ACTOR_LOCK);

      //Now delete the Actor
      DeleteSupport.getInstance().deleteActor(alifc, actor);
      return;
   }
}
```

5. Save the project. For more information, see Saving a Project [p. 12].

Packages

With the Optimal Trace Plug-in API, you can create, modify, and delete packages. You can also create, modify, and delete custom properties of a package.

This section provides specific instructions for creating and deleting packages. Additionally, this section contains instructions for creating, modifying, and deleting custom properties of a package.

Creating a Package

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Get the highest usecase package which is always the `Requirements` package of the project object.

```
//using the project object, get the package and assign it to the package object
 UseCasePackageIfc pkg = pifc.getUseCasePackage();
```

3. Now create a new package and assign a name.

```
//Only if the tryGlobalLock of projectContainer object is true, proceed.
if (ProjectContainer.getInstance().tryGlobalLock())
{
      //Place a lock on the parent package
      LockSupport.lock(pifc,pkg );

      //Create a new empty package using the PackageFactory object
      UseCasePackageIfc newPkg =
      UseCasePackageFactory.getInstance().createNewUseCasePackage(pkg);

      //Assign a name
      newPkg.setName("My New Package");

      //Add a local lock for the new package
      LockSupport.addLocallyLockedBizObject(newPkg);

      //Release the global lock using the projectContainer object
      ProjectContainer.getInstance().releaseGlobalLock();
}
```

4. Save the project. For more information, see Saving a Project [p. 12].

Deleting a Package

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Iterate through the list of packages, find the required package, and then delete it.

```
//should proceed only if the tryGlobalLock of projectContainer object is true.
if (ProjectContainer.getInstance().tryGlobalLock())
{

//Get the list of packages under the project
java.util.List pkgList = project.getAllUseCasePackages();

//find the required package
     for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
     {
     UseCasePackageIfc pkg = (UseCasePackageIfc)iterator.next();
          if(pkg.getName().equals("My Package"))
          {
          //Place a lock of type parent and all children.
          LockSupport.lock(project, pkg, LockSupport.
          PARENT_AND_ALL_INWARD_OBJECTS_LOCK_FOR_DELETE);
          //using the support class delete the package
          DeleteSupport.getInstance().deleteUseCasePackage(pkg, false);
          }
     }
//Now make sure to release the global lock using the projectContainer object
ProjectContainer.getInstance().releaseGlobalLock();
}
```

3. Save the project. For more information, see Saving a Project [p. 12].

Creating a Custom Property for a Package

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Iterate through the list of packages for the selected project. Find the desired package and create a custom property.

```
//Get the list of packages under the project
java.util.List pkgList = project.getAllUseCasePackages();

//find the required package
   for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
   UseCasePackageIfc pkg = (UseCasePackageIfc)iterator.next();
   if(pkg.getName().equals("My Package"))
   {
     //Always shout set the Inherit CP Templates to false, before creating a CP.
     pkg.setInheritCPTemplatesMap(false);

     //Get the list of CP for the package.
    CustomPropertyTemplateListIfc templatebList = pkg.getCustomPropertyTemplates();


     //Place a lock of type list and holders for the templatelist object
     LockSupport.lock(project, templatebList,
     LockSupport.CUSTOM_PROPERTY_TEMPLATE_LIST_AND_HOLDERS);

     //Create new and pass the list under which it will create.
     CustomPropertyTemplateIfc cp = CustomPropertySupport.getInstance().
     createCustomPropertyTemplate(templatebList);

     //set name and description
```

```
    cp.setName(“New CP”);
    cp.setDefinition("New CP description");
  }
}
```

3. Save the project. For more information, see Saving a Project [p. 12].

Modifying a Custom Property of a Package

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Get the desired package.

```
//Get the package
UseCasePackageIfc pkg ;

//Get the list of packages under the project
java.util.List pkgList = project.getAllUseCasePackages();

//find the required package
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
   UseCasePackageIfc pkgReq = (UseCasePackageIfc)iterator.next();
   if(pkg.getName().equals("My Package"))
   {
       pkg = pkgReq;
   }
}
```

3. Get the desired custom property by iterating and placing a lock before modifying it.

```
 //Get the list of custom properties from the current package.
CustomPropertyTemplateListIfc pkgList = pkg.getCustomPropertyTemplates();
for (Iterator iterator = pkgList.getCustomPropertyTemplates().iterator();
iterator.hasNext();)
{
     CustomPropertyTemplateIfc cptemplate = (CustomPropertyTemplateIfc)
     iterator.next();

     //Find the match
     if(cptemplate.getName().equalsIgnoreCase("My Custom Property"))
     {
         //Place a lock of type list and holders.
         LockSupport.lock(pifc,attribList,
         LockSupport.CUSTOM_PROPERTY_TEMPLATE_LIST_AND_HOLDERS);

         //Set the definition of the custom property to new value
         cptemplate.setDefinition("new updated value");
         return;
     }
}
```

4. Save the project. For more information, see Saving a Project [p. 12].

Deleting a Custom Property of a Package

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Get the desired package.

```
//Get the package
UseCasePackageIfc pkg ;

//Get the list of packages under the project
java.util.List pkgList = project.getAllUseCasePackages();

//Find the required package
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
   UseCasePackageIfc pkgReq = (UseCasePackageIfc)iterator.next();
   if(pkg.getName().equals("My Package"))
      {
          pkg = pkgReq;
      }
}
```

3. Get the desired custom property by iterating and placing a lock before deleting it.

```
 //Get the list of custom properties from the current package.
CustomPropertyTemplateListIfc pkgList = pkg.getCustomPropertyTemplates();
for (Iterator iterator = pkgList.getCustomPropertyTemplates().iterator();
iterator.hasNext();)
{
     CustomPropertyTemplateIfc cptemplate = (CustomPropertyTemplateIfc)
     iterator.next();
     //Find the match
     if(cptemplate.getName().equalsIgnoreCase("My Custom Property"))
     {
         //Place a lock of type list and holders.
         LockSupport.lock(pifc,attribList ,
         LockSupport.CUSTOM_PROPERTY_TEMPLATE_LIST_AND_HOLDERS);

         //Delete the custom property using the support class
         CustomPropertySupport.getInstance().deleteCustomPropertyTemplate
         (cptemplate, pkgList);
         return;
     }
}
```

4. Save the project. For more information, see Saving a Project [p. 12].

Simple Requirements

With the Optimal Trace Plug-in API, you can create, modify, and delete simple requirements. You can also create, modify, and delete custom properties of a simple requirement.

This section provides specific instructions for creating, modifying, and deleting simple requirements. Additionally, this section provides instructions for modifying a custom property of a simple requirement.

Creating a Simple Requirement

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Iterate through the list of packages and find the package in which to create the desired simple requirement.

```
//Proceed only if the tryGlobalLock of projectContainer object is true.
if (ProjectContainer.getInstance().tryGlobalLock())
{
```

```
    //Get the list of packages under the project
    java.util.List pkgList = project.getAllUseCasePackages();

    //Find the required package
    for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
    {
       UseCasePackageIfc pkg = (UseCasePackageIfc)iterator.next();
       if(pkg.getName().equals("My Package"))
       {
          //Place a lock on the package
          LockSupport.lock(project, pkg);

          //Create an empty simple requirement
          SimpleRequirementIfc simpleRequirement =
          SimpleRequirementFactory.getInstance().createNewSimpleRequirement(pkg);

         //Set the name of the requirement
         simpleRequirement.setName("My Simple Requirement");
       }
    }

//Release the global lock using the projectContainer object
ProjectContainer.getInstance().releaseGlobalLock();
}
```

3. Save the project. For more information, see Saving a Project [p. 12].

Modifying a Simple Requirement

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Iterate through the list of packages and find the package in which to modify the desired simple requirement.

```
//Get the list of packages under the project
java.util.List pkgList = project.getAllUseCasePackages();

//Find the required package
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
    UseCasePackageIfc pkg = (UseCasePackageIfc)iterator.next();
    //Find the required package
    if(pkg.getName().equals("My Package"))
    {
        //Get the list of requirements and find the required one
        java.util.List srList = pkg.getAllSimpleRequirementsInPackages();
        for(Iterator iteratorSr = srList.iterator();iteratorSr.hasNext();)
    {
        //Match the name and update the requirement
        SimpleRequirementIfc srReq = (SimpleRequirementIfc)iteratorSr.next();
        if(srReq.getName().equals("My Simple Requirement"))
    {
        //Update the name of the requirement
         srReq.setName("My Simple Requirement Name");
    }
 }
```

3. Save the project. For more information, see Saving a Project [p. 12].

Deleting a Simple Requirement

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Iterate through the list of packages and find the package in which to delete the desired simple requirement.

```
//Proceed only if the tryGlobalLock of projectContainer object is true
if (ProjectContainer.getInstance().tryGlobalLock())
{
    //Get the list of packages under the project
    java.util.List pkgList = project.getAllUseCasePackages();

    //Iterate through the packages of the selected project
    for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
    {
        UseCasePackageIfc pkg = (UseCasePackageIfc)iterator.next();

        //Find the required package
        if(pkg.getName().equals("My Package"))
        {
            //Get the list of requirements and find the required one
            java.util.List srList = pkg. getSimpleRequirements ();
            for(Iterator iteratorSr = srList.iterator();iteratorSr.hasNext();)
            {
                //Match the name and delete the requirement
             SimpleRequirementIfc srReq = (SimpleRequirementIfc)iteratorSr.next();

                if(srReq.getName().equals("My Simple Requirement"))
                {
                   //Start transaction for the locking
                   LockSupport.startTransaction(project);

                   //Lock the package of type simple lock
                   LockSupport.lock(project, pkg, LockSupport.SIMPLE_LOCK);

                   //If applicable, delete the refined use cases
                   Boolean deleteRefinedUseCases = true;
                   if (deleteRefinedUseCases)
                   {
                      //Place a lock on all reachable objects
                      LockSupport.lock(project, pkg, LockSupport.
                      PARENT_AND_ALL_REACHABLE_OBJECTS_LOCK_FOR_DELETE);
                   }
                   else
                   {
                      //Place a lock on the requirement of type delete.
                      LockSupport.lock(project, pkg, LockSupport.
                      PARENT_AND_ALL_INWARD_OBJECTS_LOCK_FOR_DELETE);
                   }

                   //Commit the lock transaction
                   LockSupport.commit();

                   //Delete the requirement, using the DeleteSupport object.
                   DeleteSupport.getInstance().deleteSimpleRequirement(pkg,
                   srReq, deleteRefinedUseCases);
                }
            }
        }
    }

//Now make sure to release the global lock using the projectContainer object
ProjectContainer.getInstance().releaseGlobalLock();

}
```

3. Save the project. For more information, see Saving a Project [p. 12].

Modifying a Custom Property of a Simple Requirement

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Find the package, simple requirement, and desired custom property to modify.

```
//Loop through the packages and find the required package
java.util.List pkgList = project.getAllUseCasePackages();
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
   UseCasePackageIfc pkg = (UseCasePackageIfc)iterator.next();
   if(pkg.getName().equals("My Package"))
   {
      //Find the requirement that has the custom properties
      java.util.List srList = pkg.getAllSimpleRequirementsInPackages();
      for(Iterator iteratorSr = srList.iterator();iteratorSr.hasNext();)
      {
         SimpleRequirementIfc srReq = (SimpleRequirementIfc)iteratorSr.next();
         if(srReq.getName().equals("My Requirement"))
         {
             CustomPropertyTemplateListIfc templatebList =
             srReq.getCustomPropertyTemplates();

             //Find the custom property that need to be updated
             for (Iterator iterator1 = templatebList.getCustomPropertyTemplates().
             iterator() ; iterator1.hasNext();)
             {
                CustomPropertyTemplateIfc cptemplate = (CustomPropertyTemplateIfc)

                 iterator1.next();
                 if(cptemplate.getName().equals("My Custom Property") )
                 {
                     //Place a lock of type list and holders
                     LockSupport.lock(project,templatebList , LockSupport.
                     CUSTOM_PROPERTY_TEMPLATE_LIST_AND_HOLDERS);

                  //Update the value of a custom property
                  cptemplate.setDefinition("Updated New Value");
                 }
              }
          }
      }
   }
}
```

3. Save the project. For more information, see Saving a Project [p. 12].

Items

With the Optimal Trace Plug-in API, you can create, modify, and delete an item of simple requirement. Additionally, you can create, modify, and delete custom properties of items.

This section provides specific instructions for creating and deleting an item of a simple requirement.

Creating an Item

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();
```

```
//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Place a global lock on the selected project.

```
//Proceed only if true.
if (!ProjectContainer.getInstance().tryGlobalLock())
{
 return;
}
```

3. Find the package and simple requirement in which to create an item.

```
//Loop through the packages and find the required package
java.util.List pkgList = project.getAllUseCasePackages();
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
   UseCasePackageIfc pkg = (UseCasePackageIfc)iterator.next();
   if(pkg.getName().equals("My Package"))
   {
      //Find the requirement that has the custom properties
      java.util.List srList = pkg.getAllSimpleRequirementsInPackages();
      for(Iterator iteratorSr = srList.iterator();iteratorSr.hasNext();)
      {
         SimpleRequirementIfc srReq = (SimpleRequirementIfc)iteratorSr.next();
         if(srReq.getName().equals("My Requirement"))
         {
           //place a lock on the requirement of type usecase lock
           LockSupport.lock(project, (AbstractRequirementIfc)srReq,
           LockSupport.USECase_LOCK);

           //since this is the first item, create a item list object
          ItemListIfc ilst = ItemListFactory.getInstance().createNewItemList(srReq);


           //now create a item using ItemFactory object
          AbstractStepIfc newStep = ItemFactory.getInstance().createNewItem(ilst);


           //Set name and description of the item
           newStep.setName(“New Step”);
           newStep.setDescription("New Step Description");
         }
      }
   }
}
```

4. Release the global lock.

```
//Now make sure to release the global lock using the projectContainer object
ProjectContainer.getInstance().releaseGlobalLock();
```

5. Save the project. For more information, see Saving a Project [p. 12].

Deleting an Item

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Place a global lock on the selected project.

```
//Proceed only if true.
if (!ProjectContainer.getInstance().tryGlobalLock())
{
 return;
}
```

3. Find the package and simple requirement, and then delete the desired item.

```
//Loop through the packages and find the required package
java.util.List pkgList = project.getAllUseCasePackages();
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
   UseCasePackageIfc pkg = (UseCasePackageIfc)iterator.next();
   if(pkg.getName().equals("My Package"))
   {
      //Find the requirement that has the items.
      java.util.List srList = pkg.getSimpleRequirements();
      for(Iterator iteratorSr = srList.iterator();iteratorSr.hasNext();)
      {
         SimpleRequirementIfc srReq = (SimpleRequirementIfc)iteratorSr.next();
         if(srReq.getName().equals(“Requirement Name”))
         {
             //Place a lock on the requirement
             LockSupport.lock(project, (AbstractRequirementIfc)srReq,
             LockSupport.USECase_LOCK);

             //Get the item that need delete
             java.util.List ilst = (java.util.List)srReq.getItemList().getItems();
             for(Iterator iteratorItem = ilst.iterator();iteratorItem.hasNext();)
             {
                 ItemIfc item = (ItemIfc) iteratorItem.next();
                 if(item.getName().equals(“Item Name”))
                 {
                    //Lock all the reachable objects
                    LockSupport.lock(project, srReq,
                    LockSupport.ALL_REACHABLE_OBJECTS_LOCK);

                    //pass in package and item. The Boolean parameter is to
                    //delete depended objects such as links and refinements
                    DeleteSupport.getInstance().deleteStep(pkg, item, true);
                 }
             }
          }
      }
   }
}
```

4. Release the global lock.

```
//Now make sure to release the global lock using the projectContainer object
ProjectContainer.getInstance().releaseGlobalLock();
```

5. Save the project. For more information, see Saving a Project [p. 12].

Structured Requirements

With the Optimal Trace Plug-in API, you can create, modify, and delete structured requirements. Additionally, you can create, modify, and delete custom properties of structured requirements.

This section provides specific instructions for creating and deleting structured requirements.

Creating a Structured Requirement

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Place a global lock on the selected project.

```
//Proceed only if true.
if (!ProjectContainer.getInstance().tryGlobalLock())
{
```

```
 return;
}
```

3. Find the package in which to create a structured requirement.

```
//Get the list of packages under the project
java.util.List pkgList = project.getAllUseCasePackages();

//find the required package
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
UseCasePackageIfc pkg = (UseCasePackageIfc)iterator.next();
   if(pkg.getName().equals("My Package"))
   {
      //Place a lock on the package
      LockSupport.lock(project, pkg);

      //Create new structured requirement
      UseCaseIfc strRequirement =
      UseCaseFactory.getInstance().createNewUseCase(pkg);

      //Set name of the requirement
      strRequirement.setName("My Simple Requirement");
      String priority =
      ProjectContainer.getInstance().getToolSettings().
      getDefaultUseCasePriority();

      //Priority and goal levels are mandatory
      strRequirement.setPriority(priority);

      //get goal level from projectcontainer object, which is in memory
      GoalLevelsIfc goalLevels =
      ProjectContainer.getInstance().getSelectedProject().getGoalLevels();
      GoalLevelIfc goalLevel = goalLevels.getDefaultValue();

      //assign goal level
      strRequirement.setGoalLevel(goalLevel);
   }
}
```

4. Release the global lock.

```
//Now make sure to release the global lock using the projectContainer object
ProjectContainer.getInstance().releaseGlobalLock();
```

5. Save the project. For more information, see Saving a Project [p. 12].

Deleting a Structured Requirement

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Find the package in which to delete the desired structured requirement.

```
//Proceed only if the tryGlobalLock of projectContainer object is true
if (ProjectContainer.getInstance().tryGlobalLock())
{
   //Get the list of packages under the project
   java.util.List pkgList = project.getAllUseCasePackages();

   //find the required package
   for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
   {
       UseCasePackageIfc pkg = (UseCasePackageIfc)iterator.next();

      //Find the required package
      if(pkg.getName().equals("My Package"))
```

```
      {
          //Get the list of requirements and find the required one
          java.util.List srList = pkg. getUseCases();

for(Iterator iteratorSr = srList.iterator();iteratorSr.hasNext();)
{
   //match the name and delete the requirement
   UseCaseIfc srReq = (UseCaseIfc)iteratorSr.next();
   if(srReq.getName().equals("My Simple Requirement"))
   {
      //Start transaction for the locking
      LockSupport.startTransaction(project);

      //Lock the package of type simple lock
      LockSupport.lock(project, pkg, LockSupport.SIMPLE_LOCK);

      //If applicable, delete the refined use cases for the requirement
      Boolean deleteRefinedUseCases = true;
      if (deleteRefinedUseCases)
      {
         //Place a lock on all reachable objects with help of lock support class
         LockSupport.lock(project, pkg, LockSupport.
         PARENT_AND_ALL_REACHABLE_OBJECTS_LOCK_FOR_DELETE);
      }
      else
      {
         //Place a lock on the requirement of type delete.
         LockSupport.lock(project, pkg, LockSupport.
         PARENT_AND_ALL_INWARD_OBJECTS_LOCK_FOR_DELETE);
      }

      //Commit lock transaction
      LockSupport.commit();

      //Delete the requirement, using the DeleteSupport object.
     DeleteSupport.getInstance().deleteRequirement(pkg,srReq, deleteRefinedUseCases);

      }
   }
}
}

//Now make sure to release the global lock using the projectContainer object
ProjectContainer.getInstance().releaseGlobalLock();
}
```

3. Release the global lock.

```
//Now make sure to release the global lock using the projectContainer object
ProjectContainer.getInstance().releaseGlobalLock();
```

4. Save the project. For more information, see Saving a Project [p. 12].

Steps

With the Optimal Trace Plug-in API, you can create, modify, and delete steps of a structured requirement. Additionally, you can create, modify, and delete custom properties of steps of a structured requirement.

This section provides specific instructions for creating and deleting simple requirements.

Creating a Step

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();
```

```
//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Place a global lock on the selected project.

```
//Proceed only if true
if (!ProjectContainer.getInstance().tryGlobalLock())
{
 return;
}
```

3. Find the package an structured requirement in which to create a step.

```
//Loop through the packages and find the required package
java.util.List pkgList = project.getAllUseCasePackages();
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
   UseCasePackageIfc pkg = (UseCasePackageIfc)iterator.next();
   if(pkg.getName().equals("My Package"))
   {
      //Find the requirement that has the custom properties
      java.util.List srList = pkg.getRequirements();
      for(Iterator iteratorSr = srList.iterator();iteratorSr.hasNext();)
       {
          UseCaseIfc srReq = (UseCaseIfc)iteratorSr.next();
          if(srReq.getName().equals("My Requirement"))
          {
             //Place a lock on the requirement of type usecase lock
             LockSupport.lock(project, (AbstractRequirementIfc)srReq,
             LockSupport.USECase_LOCK);

             //Get the main success scenario
             ScenarioIfc scenario = srReq.getMainSuccessScenario();

             //Create a step under the scenario
             StepIfc newStep = StepFactory.getInstance().createNewStep(scenario);

             //Assign the name to the newly created step
             newStep.setName(“My 1st Step”);
          }
      }
   }
}
```

4. Release the global lock.

```
//Release the global lock using the projectContainer object
ProjectContainer.getInstance().releaseGlobalLock();
```

5. Save the project. For more information, see Saving a Project [p. 12].

Deleting a Step

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Place a global lock on the selected project.

```
//Proceed only if true
if (!ProjectContainer.getInstance().tryGlobalLock())
{
 return;
}
```

3. Find the package an structured requirement in which to create a step.

```
//Loop through the packages and find the required package
java.util.List pkgList = project.getAllUseCasePackages();
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
   UseCasePackageIfc pkg = (UseCasePackageIfc)iterator.next();
   if(pkg.getName().equals("My Package"))
   {
      //Find the requirement that has the step
      java.util.List srList = pkg.getRequirements();
      for(Iterator iteratorSr = srList.iterator();iteratorSr.hasNext();)
      {
         //Find the requirement under which the step is present
         UseCaseIfc srReq = (UseCaseIfc)iteratorSr.next();
         if(srReq.getName().equals("Requirement Name"))
         {
            //Place a lock on requirement of type use case lock
            LockSupport.lock(project, (AbstractRequirementIfc)srReq,
            LockSupport.USECase_LOCK);

            //Get the Main Success Scenario
            MainSuccessScenarioIfc msScenario = srReq.getMainSuccessScenario();

            //Get list of steps under the Scenario
            java.util.List ilst = (java.util.List)msScenario.getSteps();
            for(Iterator iteratorItem = ilst.iterator();iteratorItem.hasNext();)
            {
               StepIfc step = (StepIfc) iteratorItem.next();

               //Get the required step
               if(step.getName().equals("My Item Name"))
               {
                   //To delete all the refinements and branches,
                   //place a lock on all reachable objects
                   LockSupport.lock(project, srReq, LockSupport.
                   ALL_REACHABLE_OBJECTS_LOCK);

                  //Use the DeleteSupport calls to delete the desired step
                  DeleteSupport.getInstance().deleteStep(pkg, step, true);
               }
            }
       }
   }
}
```

4. Release the global lock.

```
//Now make sure to release the global lock using the projectContainer object
ProjectContainer.getInstance().releaseGlobalLock();
```

5. Save the project. For more information, see Saving a Project [p. 12].

Alternate scenarios

With the Optimal Trace Plug-in API, you can create, modify, and delete alternate scenarios. Additionally, you can create, modify, and delete custom properties of alternate scenarios.

This section provides specific instructions for creating and deleting alternate scenarios.

Creating an Alternate Scenario

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Place a global lock on the selected project.

```
//Proceed only if true
if (!ProjectContainer.getInstance().tryGlobalLock())
{
 return;
}
```

3. Find the package and requirement in which to create an alternate scenario.

```
//Get the list of packages
java.util.List pkgList = project.getAllUseCasePackages();
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
   //Match the package name
   UseCasePackageIfc pkg = (UseCasePackageIfc)iterator.next();
   if(pkg.getName().equals("My Package"))
   {
      //List of requirements
      java.util.List srList = pkg.getUseCases();
      for(Iterator iteratorSr = srList.iterator();iteratorSr.hasNext();)
      {
         //Match the name of the requirement
         UseCaseIfc srReq = (UseCaseIfc)iteratorSr.next();
         if(srReq.getName().equals("My Simple Requirement"))
         {
            //Place a lock on the requirement of type use case
            LockSupport.lock(project, srReq, LockSupport.USECase_LOCK);

            //Create a new alternate scenario and pass the requirement
            AlternativeScenarioIfc newAltScenario =
            AlternativeScenarioFactory.getInstance()
            .createNewAlternativeScenario(srReq);

            //Set the name of the alternate scenario
            newAltScenario.setName(altSName);
         }
      }
   }
}
```

4. Release the global lock.

```
//Now make sure to release the global lock using the projectContainer object
ProjectContainer.getInstance().releaseGlobalLock();
```

5. Save the project. For more information, see Saving a Project [p. 12].

Deleting an Alternate Scenario

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Place a global lock on the selected project.

```
//Proceed only if true
if (!ProjectContainer.getInstance().tryGlobalLock())
{
 return;
}
```

3. Find the package and requirement in which to delete the desired alternate scenario.

```
//Get the list of packages
java.util.List pkgList = project.getAllUseCasePackages();
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
      //Match the package name
      UseCasePackageIfc pkg = (UseCasePackageIfc)iterator.next();
```

```
      if(pkg.getName().equals("My Package"))
      {
         //List of requirements
         java.util.List srList = pkg.getUseCases();
         for(Iterator iteratorSr = srList.iterator();iteratorSr.hasNext();)
         {
             //Match the name of the requirement
            UseCaseIfc srReq = (UseCaseIfc)iteratorSr.next();
            if(srReq.getName().equals("My Simple Requirement"))
            {
               //Get List of alternate scenarios
               java.util.List asList = srReq.getAlternativeScenarios();
               for(Iterator iteratorAs = asList.iterator();iteratorAs.hasNext();)
               {

                  //Match the alternate scenario and delete it
                  AlternativeScenarioIfc as =
                  (AlternativeScenarioIfc)iteratorAs.next();
                  if(as.getName().equals(altSName))
                  {
                     //Lock the requirement and of type all reachable objects
                     //Since we are deleting all the mapped entities
                     LockSupport.lock(project, srReq,
                     LockSupport.ALL_REACHABLE_OBJECTS_LOCK);

                     //Use the DeleteSupport object to delete the scenario
                     //Pass true to delete all the mapped entites
                     DeleteSupport.getInstance().deleteScenario(pkg, as, true);
                  }
               }
            }
         }
      }
}
```

4. Release the global lock.

```
//Now make sure to release the global lock using the projectContainer object
ProjectContainer.getInstance().releaseGlobalLock();
```

5. Save the project. For more information, see Saving a Project [p. 12].

Refinements

With the Optimal Trace Plug-in API, you can create, modify, and delete refinements. Additionally, you can create, modify, and delete custom properties of refinements.

This section provides specific instructions for creating and deleting refinements.

Creating a Refinement

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Get the package, simple requirement, and step in which to create the refinement.

```
//Get the list of packages
java.util.List pkgList = project.getAllUseCasePackages();
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
      //Match the package name
      UseCasePackageIfc pkg;
      UseCasePackageIfc tpkg = (UseCasePackageIfc)iterator.next();
      if(pkg.getName().equals("My Package"))
      {
```

```
            pkg = tPkg;
        }
}
//Get the structured requirement by calling getUseCase method of package object
UseCaseIfc fromReq = pkg.getUseCase(fromReqName);

//Get the scenario
ScenarioIfc sce = fromReq.getMainSuccessScenario();

//By using the Scenario object, get the step
StepIfc fromStep = (StepIfc) sce.getChild(fromStepName, Step.class);
```

3. Create an empty structured requirement in the package

```
//Place a lock on the package
 LockSupport.lock(project, pkg);

//Create new structured requirement
UseCaseIfc strRequirement =
UseCaseFactory.getInstance().createNewUseCase(pkg);

//Set the name of the requirement
strRequirement.setName("My Simple Requirement");
String priority =
ProjectContainer.getInstance().getToolSettings().getDefaultUseCasePriority();

//Set the mandatory priority and goal levels
strRequirement.setPriority(priority);

//Get goal level from ProjectContainer object, which is in memory
GoalLevelsIfc goalLevels = ProjectContainer.getInstance().
getSelectedProject().getGoalLevels();
GoalLevelIfc goalLevel = goalLevels.getDefaultValue();

//Assign a goal level
strRequirement.setGoalLevel(goalLevel);
```

4. Create a new refinement.

```
//Place a lock of type use case lock on the requirement.
LockSupport.lock(project,fromReq , LockSupport.USECase_LOCK);

//Create new refinement and pass the step and newly created  requirement

RefinementIfc refinement =
RefinementFactory.getInstance().createNewRefinement(fromStep, strRequirement);
```

5. Save the project. For more information, see Saving a Project [p. 12].

Deleting a Refinement

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Get the package, structured requirement, and step in which to delete the desired refinement.

```
//Get the list of packages
java.util.List pkgList = project.getAllUseCasePackages();
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
     //Match the package name
     UseCasePackageIfc pkg;
        UseCasePackageIfc tpkg = (UseCasePackageIfc)iterator.next();
        if(pkg.getName().equals("My Package"))
        {
            pkg = tPkg;
        }
```

```
}

//Get the structured requirement by calling the getUsecase method of package oject
UseCaseIfc fromReq = pkg.getUseCase(fromReqName);

//Get the requirement that the refinement is pointing to
UseCaseIfc toReq = pkg.getUseCase(toReqName);

//Get the scenario
ScenarioIfc sce = fromReq.getMainSuccessScenario();

//By using the Scenario object, get the step
StepIfc fromStep = (StepIfc) sce.getChild(fromStepName, Step.class);
```

3. Find the refinement to delete by matching the name of the requirement to which the particular refinement is pointing.

```
//Get list of available refinements
java.util.List lstRefinements = fromStep.getRefinements();
for (Iterator iteratorRef = lstRefinements.iterator(); iteratorRef.hasNext();)
{
   RefinementIfc ref = (RefinementIfc) iteratorRef.next();
   AbstractRequirementIfc absToUseCase = ref.getRefinedToUseCase();

   //Check to see if this is the refinement we like to delete
   if(absToUseCase.equals(toReq))
   {
      // Place a lock of type all reachable objects
      LockSupport.lock(project, fromReq, LockSupport.ALL_REACHABLE_OBJECTS_LOCK);

      //Delete refinement and all the refined requirements too by passing true
      DeleteSupport.getInstance().deleteRefinement(ref,true);
   }
}
```

4. Save the project. For more information, see Saving a Project [p. 12].

Links

With the Optimal Trace Plug-in API, you can create, modify, and delete links. Additionally, you can create, modify, and delete custom properties of links.

This section provides specific instructions for creating and deleting links.

Creating a Link

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Get the package, simple requirement, and item in which to create link.

```
//Get List of packages
java.util.List pkgList = project.getAllUseCasePackages();
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
      //Match the package name
      UseCasePackageIfc pkg;
      UseCasePackageIfc tpkg = (UseCasePackageIfc)iterator.next();
      if(pkg.getName().equals("My Package"))
      {
         pkg = tPkg;
      }
}
```

3. Find the desired simple requirement and step, and then create a Bucket List and the new link under it.

```
java.util.List ilst = (java.util.List)pkg.getSimpleRequirement("My
Requirement").getItemList().getItems();

//Loop through the items and find the match
for(Iterator iteratorItem = ilst.iterator();iteratorItem.hasNext();)
{
   ItemIfc item = (ItemIfc) iteratorItem.next();
   if(item.getName().equals("My Item Name"))
   {
      //Place a lock on the item
      LockSupport.lock(project,item);

      //Create a new link
      ExternalLinkIfc newLink = null;
      newLink = ExternalLinkFactory.getInstance().createNewExternalLink(item);

      //Create a new link bucket to hold the links
      LinkBucketIfc linkBucketIfc = LinkBucketFactory.getInstance().
      createNewLinkBucket(item.getName(),item);

      //Set the link bucket to the item
      item.setLinkBucket(linkBucketIfc);
      linkBucketIfc.addExternalLink(newLink);

      //Assign name and link location
      newLink.setName(linkName);
      newLink.setLinkValue(linkLocation);
   }
}
```

4. Save the project. For more information, see Saving a Project [p. 12].

Deleting a Link

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Get the package in which the link appears.

```
//Get List of packages
java.util.List pkgList = project.getAllUseCasePackages();
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
      //Match the package name
      UseCasePackageIfc pkg;
      UseCasePackageIfc tpkg = (UseCasePackageIfc)iterator.next();
      if(pkg.getName().equals("My Package"))
      {
         pkg = tPkg;
      }
}
```

3. Find the simple requirement and step in which the link appears, and tehn , and then delete it.

```
java.util.List ilst = (java.util.List)pkg.getSimpleRequirement("My
Requirement").getItemList().getItems();

//Loop through the items and find the match
for(Iterator iteratorItem = ilst.iterator();iteratorItem.hasNext();)
{
   ItemIfc item = (ItemIfc) iteratorItem.next();
   if(item.getName().equals("My Item Name"))
```

```
   {
      //Get list of links under the item
      ExternalLinkIfc newLink = null;
      java.util.List llinks = item.getLinkBucket().getExternalLinks();
      for(Iterator iteratorLink = llinks.iterator();iteratorLink.hasNext();)
      {
         //Find the required link to delete
         ExternalLinkIfc link = (ExternalLinkIfc) iteratorLink.next();
         if(link.getName().equals(linkName))
         {
            //With help of delete support class, delete the link
            //Place a lock on the item.
            LockSupport.lock(project,item);
            DeleteSupport.getInstance().deleteLink(item.getLinkBucket(), link);
         }
      }
   }
}
```

4. Save the project. For more information, see Saving a Project [p. 12].

Notes

With the Optimal Trace Plug-in API, you can create, modify, and delete notes. Additionally, you can create, modify, and delete custom properties of notes.

This section provides specific instructions for creating and deleting notes.

Creating a Note

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Place a lock on the project and create a new note.

```
//Place a lock on the project
LockSupport.lock(project, project);

//Create a new note by passing the project's NoteHolder object.
NoteIfc newNote = NoteFactory.getInstance().createNewNote((NoteHolderIfc)project);

//Set the note name and note subject
newNote.setName("My Note Name");
newNote.setSubject(noteSubject);
```

3. Save the project. For more information, see Saving a Project [p. 12].

Deleting a Note

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Get the list of available notes from the selected project. Find the note to delete, place a lock on the project, and then delete the note.

```
//Get the NoteBucket object of the project
 NoteBucketIfc nbi = project.getNoteBucket();
```

```
//Get list of available notes
java.util.List notes = nbi.getNotes();

//Loop through and find the note to delete
for (Iterator iterator = notes.iterator(); iterator.hasNext();)
{
   NoteIfc note = (NoteIfc) iterator.next();
   if(note.getName().equals("My Note Name"))
   {
      //Place a lock on the project
      LockSupport.lock(project, project);

      //Using delete support class delete the note by passing NoteBucket and Note
      DeleteSupport.getInstance().deleteNote(nbi, note);
   }
}
```

3. Save the project. For more information, see Saving a Project [p. 12].

Branches

With the Optimal Trace Plug-in API, you can create, modify, and delete branches. Additionally, you can create, modify, and delete custom properties of branches.

This section provides specific instructions for creating and deleting branches.

Creating a Branch

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Place a global lock on the project.

```
//Proceed only if true
if (!ProjectContainer.getInstance().tryGlobalLock())
{
   return;
}
```

3. Get the package in which to create the branch.

```
//Get the list of packages
java.util.List pkgList = project.getAllUseCasePackages();
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
   //Match the package name
   UseCasePackageIfc pkg;
   UseCasePackageIfc tpkg = (UseCasePackageIfc)iterator.next();
   if(pkg.getName().equals("My Package"))
   {
      pkg = tPkg;
   }
}
```

4. Get the *from* requirement, *to* requirement, and the step, and then create the branch

```
 //Get the from requirement of the package
 UseCaseIfc fromReq = pkg.getUseCase(fromReqName);

//Get the MainSuccessScenario
MainSuccessScenarioIfc mss = fromReq.getMainSuccessScenario();

//Get the step in which to create a branch
StepIfc fromStep = (StepIfc) mss.getChild(fromStepName, Step.class);
```

```
//Get the to requirement
UseCaseIfc toReq = pkg.getUseCase(toReqName);

//Place a use case lock
LockSupport.lock(project, fromReq, LockSupport.USECase_LOCK);

//Create a new branch using the BranchFactory object
BranchFactory.getInstance().createNewBranch(branchName,fromStep,toReq);
```

5. Release the global lock.

```
//Now make sure to release the global lock using the projectContainer object
ProjectContainer.getInstance().releaseGlobalLock();
```

6. Save the project. For more information, see Saving a Project [p. 12].

Deleting a Branch

1. Get the selected project object.

```
//Get the instance of a project container object, which is a singleton.
ProjectContainer  projCnt =  ProjectContainer.getInstance();

//From the project container object, get the selected project.
ProjectIfc   pifc = projCnt.getSelectedProject();
```

2. Place a global lock on the project.

```
//Proceed only if true.
if (!ProjectContainer.getInstance().tryGlobalLock())
{
   return;
}
```

3. Get the package in which to create the branch.

```
//Get List of packages
java.util.List pkgList = project.getAllUseCasePackages();
for (Iterator iterator = pkgList.iterator(); iterator.hasNext();)
{
   //Match the package name
   UseCasePackageIfc pkg;
   UseCasePackageIfc tpkg = (UseCasePackageIfc)iterator.next();
   if(pkg.getName().equals("My Package"))
   {
      pkg = tPkg;
   }
}
```

4. Get the *from* requirement, *to* requirement, and the step, and then delete the branch

```
 //From package get the requirement
 UseCaseIfc fromReq = pkg.getUseCase(fromReqName);

//Get the MainSuccessScenario
MainSuccessScenarioIfc mss = fromReq.getMainSuccessScenario();

//Get the step from where to create a Branch
StepIfc fromStep = (StepIfc) mss.getChild(fromStepName, Step.class);

//Get the Branch to requirement
UseCaseIfc toReq = pkg.getUseCase(toReqName);

//Get list of available branches, find the match
java.util.List branches = fromStep.getBranches();
for (Iterator iteratorBranch = branches.iterator(); iteratorBranch.hasNext();)
{
   BranchIfc branch = (BranchIfc) iteratorBranch.next();
   if(branch.getName().equals(branchName))
   {
      //Place a lock on the requirement
```

```
      LockSupport.lock(project, fromReq, LockSupport.ALL_REACHABLE_OBJECTS_LOCK);

      //With help of delete support, delete the branch
      DeleteSupport.getInstance().deleteBranch(branch, true);
   }
}
```

5. Release the global lock.

```
//Now make sure to release the global lock using the projectContainer object
ProjectContainer.getInstance().releaseGlobalLock();
```

6. Save the project. For more information, see Saving a Project [p. 12].

Index

O