Rational Team API
Tutorial for RequisitePro

(c) Copyright IBM Corp. 2006

Legal Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department BCFB

20 Maguire Road
Lexington, MA 02421
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

(c) (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. (c) Copyright IBM Corp. _enter
the year or years_. All rights reserved.

Trademarks

http://www.ibm.com/legal/copytrade.shtml

Contents

Overview

This tutorial provides sample code examples that use the Rational Team API to implement some common RequisitePro requirements management use cases. The use cases include:

Open a project and display its properties
- Open a project
- Display project properties
- Display all of the views in the project

View project structure
- Display all requirement types in a project
- Lookup a specific requirement type
- Display the attribute definitions of a requirement type

Requirements
- Display all requirements, their children and trace-to relationships, of a particular requirement type
- Retrieve an individual requirement and display it's attributes
- Update a requirement's display name and comment
- Create a requirement in a folder
- Move a requirement to another folder
- Delete a requirement

The example begins with the following shell:

```
public class RPTutorial
{
// Main method: Execution starts here
public static void main(String[] args)
{
        System.out.println(“Starting TeamAPI example application”);
       try {
           runTutorial();
       }
       catch (Exception e) {
           e.printStackTrace();
       }
        System.out.println(“Finished TeamAPI example application”);
}
}
```

Before you can write the use case code examples, you need to import required interfaces and get a Provider object.

What interfaces to import

Get a provider

Following these steps, you can include code from one of the use case examples, or combine them, for a complete sample application.

What interfaces to import

Since the use case examples use the RequisitePro APIs within Rational Team API, you must import the RequisitePro interfaces. You can use a wildcard to have all the RequisitePro-specific APIs available:

```
import com.ibm.rational.wvcm.stp.rp.*;
```

You can add more **import** statements to your application as needed.

The entire list of import statements for all of the use cases in this tutorial requires additional classes to import. In addition to the RequisitePro APIs, the following list includes some Rational Team API and WVCM base classes and some Java utility classes:

```
// WVCM classes to import for this tutorial
import javax.wvcm.PropertyNameList;
import javax.wvcm.ProviderFactory;
import javax.wvcm.ResourceList;
import javax.wvcm.PropertyNameList.NestedPropertyName;
import javax.wvcm.PropertyNameList.PropertyName;
import javax.wvcm.ProviderFactory.Callback;
import javax.wvcm.ProviderFactory.Callback.Authentication;
import javax.wvcm.WvcmException;
// Rational Team API classes to import for this tutorial
import com.ibm.rational.wvcm.stp.Location;
import com.ibm.rational.wvcm.stp.PropertyDefinition;
import com.ibm.rational.wvcm.stp.Repository;
import com.ibm.rational.wvcm.stp.Resource;
import com.ibm.rational.wvcm.stp.Selector;
import com.ibm.rational.wvcm.stp.rp.Attribute;
import com.ibm.rational.wvcm.stp.rp.AttributeDefinition;
import com.ibm.rational.wvcm.stp.rp.Project;
import com.ibm.rational.wvcm.stp.rp.Provider;
import com.ibm.rational.wvcm.stp.rp.Relationship;
import com.ibm.rational.wvcm.stp.rp.Requirement;
import com.ibm.rational.wvcm.stp.rp.RequirementType;
import com.ibm.rational.wvcm.stp.rp.View;

private static void runTutorial() throws Exception {
```

Get a provider

The first thing a Team API application needs to do is get a Provider object. This connects the Team API interfaces to the underlying Team API Provider for access to the IBM Rational software product resources. Here is our application with added code to get a provider object:

```
    /*
     * Create an instance of a callback object that contains authentication
     * information.  This callback object will be called when a new
     * repository is accessed.
     */
    MyCallback cb = new MyCallback();

    /*
     * Initialize a provider.
```

```
 */
Provider provider = (Provider) ProviderFactory.createProvider(
        Provider.CLIENT_HOSTED_PROVIDER, cb, null);

/*
 * OPEN A PROJECT AND DISPLAY ITS PROPERTIES
 *   - Open a project
 *   - Display project properties
 *   - Display all of the views in the project
 */
```

Open a project

```
/*
 * Open a RequisitePro project, by constructing a Selector using the RQS
 * file path as name of the repository.
 */
String projectPath = "C:\\Program Files\\Rational\\RequisitePro\\" +
                "samples\\Learning_Project-Use_Cases\\LEARNING - USE CASES.RQS";
Selector selector = new Selector(
        Selector.Namespace.PROJECT,
        "",
        Repository.Type.REQUISITE_PRO,
        projectPath);

/*
 * Construct a location from the selector.
 */
Location projectLocation = provider.location(
        selector,
        false,
        Selector.Namespace.NIL);

/*
 * Use the location to create a proxy for the project resource.
 */
Project project = (Project) provider.rpProject(projectLocation);
```

Display project properties

```
/*
 * Read a set of project properties, which are specified by creating
 * a PropertyNameList from an array of PropertyNames.
 */
project = (Project) project.doReadProperties(
        new PropertyNameList(
                new PropertyName[] {
                        Resource.COMMENT,
                        Resource.DISPLAY_NAME,
                        Resource.LAST_MODIFIED,
                        Project.PREFIX
                }));
```

```
/*
 * Display the project properties
 */
System.out.println("Project properties: ");
System.out.println("  display-name: " + project.getDisplayName());
System.out.println("  comment: " + project.getComment());
System.out.println("  last-modified: " + project.getLastModified());
System.out.println("  prefix: " + project.getPrefix());
```

Display all of the views in the project

```
/*
 * Create a PropertyNameList from set of View PropertyNames.  This is
 * then used to construct a NestedPropertyName that will return all of
 * the views in the project, as well as the specified properties of
 * each view.
 */
PropertyNameList VIEW_PROPERTIES = new PropertyNameList(
        new PropertyName[] {
                Resource.DISPLAY_NAME,
                Resource.COMMENT,
                View.LAST_MODIFIED,
                View.PRIMARY_QUERY,
                View.SECONDARY_QUERY,
                View.VIEW_TYPE,
                View.VISIBILITY
        });

/*
 * Read the view properties from the project.
 */
project = (Project) project.doReadProperties(
        new PropertyNameList(
                new PropertyName[] {
                        new NestedPropertyName(
                                Project.ALL_VIEWS,
                                VIEW_PROPERTIES
                        )}));

/*
 * Iterate through the views and display properties of each view.
 */
ResourceList allViews = project.getAllViews();
System.out.println("Project views: ");
for (int i = 0; i < allViews.size(); i++) {
    View view = (View) allViews.get(i);
    System.out.println("  location: " + view.location());
    System.out.println("    display-name: " + view.getDisplayName());
    System.out.println("    comment: " + view.getComment());
    System.out.println("    last-modified: " + view.getLastModified());
    System.out.println("    view-type: " + view.getViewType());
    System.out.println("    visibility: " + view.getVisibility());
    System.out.println("    primary-query: " +
            view.getPrimaryQuery().location());
}

/*
 * VIEW PROJECT STRUCTURE
 *   - Display all requirement types in a project
 *   - Lookup a specific requirement type
 *   - Display the attribute definitions of a requirement type
 */

/*
 * RequisitePro project security (users and groups) and project
 * structure (document types, requirement types, attribute definitions)
 * is only available read-only from the Team API.  Project security and
 * project structure may not be modified.
 */
```

Display all requirement types in a project

```
/*
 * Create a NestedPropertyName, so that the requirement types may be
 * retrieved along with the properties of each requirement type.
 */
PropertyNameList REQTYPE_PROPERTIES = new PropertyNameList(
        new PropertyName[] {
                Resource.DISPLAY_NAME,
                Resource.COMMENT,
                RequirementType.MUST_CONTAIN,
                RequirementType.PREFIX
        });
/*
 * Retrieve the requirement types
 */
project = (Project) project.doReadProperties(
        new PropertyNameList(
                new PropertyName[] {
                        new NestedPropertyName(
                                Project.REQUIREMENT_TYPE_SET,
                                REQTYPE_PROPERTIES
                        )}));

/*
 * Display the properties of each requirement type.
 */
ResourceList reqTypes = project.getRequirementTypeSet();
System.out.println("Project requirement types: ");
for (int i = 0; i < reqTypes.size(); i++) {
   RequirementType reqType = (RequirementType) reqTypes.get(i);
   System.out.println("  location: " + reqType.location());
   System.out.println("  display-name: " + reqType.getDisplayName());
   System.out.println("  comment: " + reqType.getComment());
   System.out.println("  must-contain: " + reqType.getMustContain());
   System.out.println("  prefix: " + reqType.getPrefix());
}
```

Lookup a specific requirement type

```
/*
 * Display the attribute definitions and properties of a specific
 * requirement type.  A Selector is constructed using the display-name
 * of a requirement type.
 */
Selector reqTypeSelector = new Selector(
        Selector.Namespace.REQUIREMENT_TYPE,
        "Use Case",
        Repository.Type.REQUISITE_PRO,
        projectPath);

/*
 * Construct a location from the requirement type Selector.
 */
Location reqTypeLocation =
    (Location) provider.location(
            reqTypeSelector,
            false,
            Selector.Namespace.NIL);

/*
 * Obtain a requirement type proxy based upon the Location.
 */
RequirementType reqType =
    (RequirementType) provider.rpRequirementType(reqTypeLocation);
```

Display the attribute definitions of a requirement type

```
/*
 * Construct a PropertyNameList which will allow the
 * AttributeDefinitions to be obtained along with their properties.
 */
PropertyNameList FIELD_DEFINITION_PROPERTIES = new PropertyNameList(
        new PropertyName[] {
                Resource.DISPLAY_NAME,
                PropertyDefinition.VALUE_TYPE,
                AttributeDefinition.AUTO_SUSPECT,
                AttributeDefinition.DEFAULT_VALUE
        });

/*
 * Read the attribute definitions from the requirement type.
 */
reqType = (RequirementType) reqType.doReadProperties(
        new PropertyNameList(
                new PropertyName[] {
                        new NestedPropertyName(
                                RequirementType.FIELD_DEFINITIONS,
                                FIELD_DEFINITION_PROPERTIES
                        )}));
```

```
/*
 * Display the properties of the attribute definitions.
 */
ResourceList attrDefs = reqType.getFieldDefinitions();
System.out.println("Requirement type attribute definitions: ");
for (int i = 0; i < attrDefs.size(); i++) {
    AttributeDefinition attributeDef =
        (AttributeDefinition) attrDefs.get(i);

    System.out.println("  location: " + attributeDef.location());
    System.out.println("  display-name: " +
            attributeDef.getDisplayName());
    System.out.println("  value-type: " + attributeDef.getValueType());
    System.out.println("  auto-suspect: " +
            attributeDef.getAutoSuspect());
    System.out.println("  default-value: " +
            attributeDef.getDefaultValue());
}

/*
 * REQUIREMENTS
 *   - Display all requirements, their children and trace-to
 *     relationships, of a particular requirement type
 *   - Retrieve an individual requirement and display its attributes
 *   - Update a requirement's display name and comment
 *   - Create a requirement in a folder
 *   - Move a requirement to another folder
 *   - Delete a requirement
 */
```

Display all requirements, their children and trace-to relationships, of a particular requirement type

```
/*
 * Retrieve all of the requirements of this type.  Properties that
 * will be diplayed from each requirement include the location of
 * children, as well as properties of any trace-to relationships.
 */
PropertyNameList RELATIONSHIP_PROPERTIES = new PropertyNameList(
        new PropertyName[] {
                Relationship.DESTINATION_REQUIREMENT,
                Relationship.SUSPECT
        });

PropertyNameList REQUIREMENT_PROPERTIES = new PropertyNameList(
        new PropertyName[] {
                Resource.DISPLAY_NAME,
                Resource.COMMENT,
                Requirement.CHILDREN,
                Requirement.HAS_CHILDREN,
                Requirement.IS_DOCUMENT_BASED,
                Requirement.IS_MODIFIED,
                Requirement.PREFIX_TAG,
        Requirement.REQUIREMENT_TYPE,
        Requirement.REVISIONS,
        Requirement.DOCUMENT,
        Requirement.PARENT,
        new NestedPropertyName(
            Requirement.TRACES_TO,
            RELATIONSHIP_PROPERTIES
        )
    });

/*
 * Read the requirements from the requirement type.  Several properties
 * of each requirement are also being read.
 */
reqType = (RequirementType) reqType.doReadProperties(
    new PropertyNameList(
        new PropertyName[] {
            new NestedPropertyName(
                RequirementType.REQUIREMENTS,
                REQUIREMENT_PROPERTIES
            )}}));

ResourceList reqList = reqType.getRequirements();
System.out.println("Requirement type requirements: ");
for (int i = 0; i < reqList.size(); i++) {
  Requirement req = (Requirement) reqList.get(i);
  System.out.println("  location: " + req.location());
  System.out.println("    display-name: " + req.getDisplayName());
  System.out.println("    comment: " + req.getComment());
  System.out.println("    has-children: " + req.getHasChildren());
  System.out.println("    is-document-based: " +
      req.getIsDocumentBased());
  System.out.println("    is-modified: " + req.getIsModified());
  System.out.println("    prefix-tag: " + req.getPrefixTag());
  System.out.println("    requirement-type: " +
      req.getRequirementType());
  System.out.println("    document: " + req.getDocument());
  System.out.println("    parent: " + req.getParent());

  /* Display any child requirements */
  for (int j = 0; j < req.getChildren().size(); j++) {
    Relationship child = (Relationship) req.getChildren().get(j);
    System.out.println("    child location: " + child.location());
  }

  /* Display any trace-to relationships */
  for (int j = 0; j < req.getTracesTo().size(); j++) {
    Relationship trace = (Relationship) req.getTracesTo().get(j);
    System.out.println("    trace-to:");
    System.out.println("      destination-requirement: " +
        trace.getDestinationRequirement());
    System.out.println("      is-suspect: " + trace.getSuspect());
  }
}
```

Retrieve an individual requirement and display its attributes

```
/*
 * Construct a selector for some requirement using the prefix-tag.
 */
Selector reqSelector = new Selector(
        Selector.Namespace.REQUIREMENT,
        "FEAT4",
        Repository.Type.REQUISITE_PRO,
        projectPath);

Location reqLocation =
    (Location) provider.location(
            reqSelector,
            false,
            Selector.Namespace.NIL);

/* Construct a proxy for the requirement resource */
Requirement req =
    (Requirement) provider.rpRequirement(reqLocation);

/* Read the user-defined fields from the requirement.  These will be
 * to construct a list of property names for the fields.
 */
req = (Requirement) req.doReadProperties(
        new PropertyNameList(
                new PropertyName[] {
                        Requirement.FIELDS
                        }));

PropertyNameList fieldNameList = req.getFields().getPropertyNameList();
PropertyName[] fieldNames = fieldNameList.getPropertyNames();

/*
 * The field values have not been requested yet, we didn't know them at
 * the time that the doReadProperties() was last done.  Therefore,
 * request the properties again, this time providing all the field
 * names.  Note that these properties will be the same for all
 * requirements of this type.
 */
req = (Requirement) req.doReadProperties(fieldNameList);

for (int i = 0; i < fieldNames.length; i++) {
    Object realValue = null;

    /*
     * Print out value of each field using getProperty().
     */
    realValue = req.getProperty(fieldNames[i]);

    if (realValue instanceof Attribute) {
        Attribute attribute = (Attribute) realValue;
        System.out.println(attribute.getName() + ": " +
              attribute.getValue());
    }
}
```

Update a requirement's display name and comment

```
/*
 * Update a requirement
 */
/* Change the display name directly */
req.setDisplayName("This is the requirement name");

/* Change the COMMENT using the PropertyName */
PropertyNameList.PropertyName prop = Resource.COMMENT;
req.setProperty(prop, "This is the requirement text");

req.doWriteProperties();
```

Create a requirement in a folder

```
/*
 * Create a requirement.  The name segment contains the package (folder)
 * hierarchy followed by the requirement type prefix.
 */
Selector newReqSelector = new Selector(
        Selector.Namespace.REQUIREMENT,
        "Learning Project - Use Cases/Glossary/FEAT",
        Repository.Type.REQUISITE_PRO,
        projectPath);
Location newReqLocation = (Location) provider.location(
        newReqSelector,
        false,
        Selector.Namespace.NIL);

Requirement newReq =
    (Requirement) provider.rpRequirement((Location) newReqLocation);

/*
 * A requirement must have either the DISPLAY_NAME or COMMENT property
 * set.
 */
newReq.setDisplayName("This shall be a requirement");

/* Create the requirement, commit is done by default. */
newReq = (Requirement) newReq.doCreateGeneratedResource(null);

/* Retrieve the prefix-tag necessary to build the new location */
newReq = (Requirement) newReq.doReadProperties(REQUIREMENT_PROPERTIES);
```

Move a requirement to another folder

```
        /*
         * Move a requirement to another folder.
         */
        newReqSelector = new Selector(
                Selector.Namespace.REQUIREMENT,
                "Learning Project - Use Cases/Features and Vision/" +
                        newReq.getPrefixTag(),
                Repository.Type.REQUISITE_PRO,
                projectPath);

        newReqLocation = (Location) provider.location(
                newReqSelector,
                false,
                Selector.Namespace.NIL);

        req.doRebind(newReqLocation, false);
```

Delete a requirement

```
        /*
         * Delete a requirement
         */
        newReq.doUnbind();

        /*
         * Terminate the provider, closing the open project
         */
        provider.terminate();
    }
```

The authentication callback is intended to enable the RequisitePro sub-provider to authenticate a user of a client application before doing operations (such as reading the properties of a requirement or modifying attribute values). The authentication callback gives your application a chance to collect authentication information (username and password) from the user running the application, and then pass that information to the RequisitePro sub-provider. For simplicity, the example above assumes a hard-coded username. In a real application, your implementation of the Callback interface would interact with your user to collect this information (for example, by including a login window).

```
    private static class MyCallback implements Callback {

                        private static class MyAuthentication implements Authentication
                        {
                                private String m_realm;

                                public MyAuthentication(
                                                String realm)
                                {
```

```
                super();
                m_realm = realm;
            }

            public String loginName()
            {
                return "admin";
            }

            public String password()
            {
                return "";
            }
        }

        public Authentication getAuthentication(
                String realm,
                Integer retryCount) {
           return new MyAuthentication(realm);
        }
    }
}
```