QADirector

Integration and SDK Reference

Release 6.1

Micro Focus (IP) Ltd. has made every effort to ensure that this book is correct and accurate, but reserves the right to make changes without notice at its sole discretion at any time. The software described in this document is supplied under a license and may be used or copied only in accordance with the terms of such license, and in particular any warranty of fitness of Micro Focus software products for any particular purpose is expressly excluded and in no event will Micro Focus be liable for any consequential loss.

Local Build: September 9, 2009, 7:48

Contents

CHAPTER 1

Introduction

Intended Usage

This Software Development Kit, including the documentation, sample code and APIs provided within, are intended for the express purpose of developing integrations that interface with *QA*Director. Micro Focus will provide support for the *QA*Director application and associated APIs. It is the responsibility of the user to properly utilize the SDK to develop, to debug, to deploy, and to support any applications derived from its usage. It is recommended that any applications developed with this SDK be thoroughly tested in a non-production environment and all data backed up before deploying to a production environment.

NOTE

Only those *QA*Director API classes and members referenced within this document are supported.

How to Use This Reference

This reference document includes information about how to use the *QA*Director SDK/API and integration components. Specifically, it covers:

- Using the *QA*Director SDK/API defect integration to connect your defect tool to *QA*Director
- Using the *QA*Director SDK/API automated testing integration to integrate your automated testing tool with *QA*Director.
- Using the *QA*Director SDK/API classes and members to create custom applications to integrate with *QA*Director.

Related Documentation

The *QA*Director documentation set includes the following:

- The *QADirector Installation Guide* includes system requirements and instructions for installing the ***QA*Director web server**, the *QA*Director database, the **Test Management Server**, the **QADirector Integration Plug-In**, the ***QA*Director client**, **Manual Testing**, and the **Test Execution Agent**.
- The *QADirector Online Help* provides how-to, reference, and conceptual information on the *QA*Director centers, tools, procedures, and the full client application.
- The *QADirector Release Notes* contains System Requirements, Known Issues, Technical Notes, and What's New information for each release.
- The *QADirector Integration and SDK Reference* contains information about how to use the *QA*Director SDK/API and integration components.
- The *Distributed License Management Installation Guide* provides instructions for installing and configuring a license for *QA*Director.
- The *QADirector - CaliberRM Integration Help* contains reference and how-to procedures for integrating your CaliberRM requirements into *QA*Director.

Getting Help

If ever you have any problems or you would like additional technical information or advice, there are several sources. In some countries, product support from Micro Focus may be available only to customers who have maintenance agreements.

If you obtained this product directly from Micro Focus, contact us as described below. If you obtained it from another source, such as an authorized distributor, contact them for help first. If they are unable to help, contact us as described below.

However you contact us, please try to include the information below, if you have it. The more information you can give, the better Product Support can help you. But if you don't know all the answers, or you think some are irrelevant to your problem, please give whatever information you have.

- The name, release (version), and build number of the product.
- Installation information, including installed options, whether the product uses local or network databases, whether it is installed in the default directories, whether it is a standalone or network installation, and whether it is a client or server installation.
- Environment information, such as the operating system and release on which the product is installed, memory, hardware/network specifications, and the names and releases of other applications that were running.
- The location of the problem in the product software, and the actions taken before the problem occurred.
- The exact product error message, if any.
- The exact application, licensing, or operating system error messages, if any.
- Your Micro Focus client, office, or site number, if available.

Contact

Our web site gives up-to-date details of contact numbers and addresses. To connect, enter *www.microfocus.com* in your browser to go to the Micro Focus home page, or go to *http://supportline.microfocus.com*.

CHAPTER 2

Third-Party Tool Integration

Defect Integration

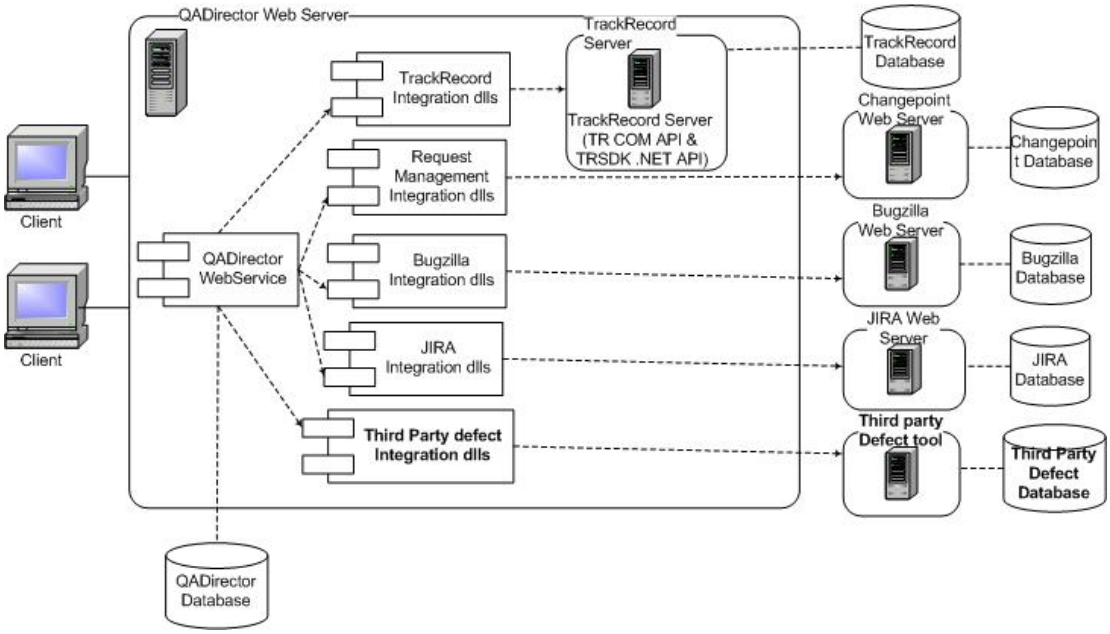
Defect tracking is an integral part of any test management cycle. *QA*Director facilitates this by providing an open integration for defect tracking tools. When a *QA*Director test fails, if the defect integration is set up, users can submit defects directly from the failed result in *QA*Director to their defect tool. The defects submitted from *QA*Director can be tracked and managed from *QA*Director's **Defect Center**.

Follow the steps in this section to use any third party defect tool with *QA*Director. See Deployment [p. 52] after your code is compiled to complete the necessary steps for your integration.

The points of integration include:

- Testing your connection.
- Retrieving defect fields.
- Retrieving defects for display in *QA*Director's **Defect Center.**
- Submitting defects.
- Editing defects.
- Launching your defect tool from *QA*Director.

The following illustration shows the *QA*Director defect tracking integration infrastructure.



System Requirements

- A defect tracking tool.
- Visual Studio 2005 and the .NET framework 2.0 for building your integration DLL.
- Refer to the defect integration C# code samples installed with this SDK for a better understanding of the integration architecture.

Code Reference

For the *QA*Director defect tracking integration, you will be creating a custom DLL to integrate *QA*Director with your defect tool. You will need to analyze the necessary APIs and integration points of the defect tool that will integrate with *QA*Director. *QA*Director requires that the DLL you create have classes named `IDefectTrackingIntegration` and `ToolClass` which follow the specific format documented in this section. When your DLL is complete, continue to the Deployment [p. 52] section.

IDefectTrackingIntegration Interface

The IDefectTrackingIntegration Interface declares the methods that need to be implemented in the integration application.

See ToolClass [p. 14] for an example of how to create and implement this interface.

ToolClass

`ToolClass` is derived from the `IDefectTrackingIntegration` interface. All of its interface methods should be implemented in this class:

```
interface IDefectTrackingIntegration
{
      string TestConnection(string inputXML);
      string SubmitDefectToTool(string inputXML);
      string GetDefectFieldListFromTool(string inputXML);
```

```
      string GetDefectListFromTool(string inputXML);
      string LaunchDefectTool(string inputXML);
      string EditDefectItem(string inputXML);
}

ToolClass:IDefectTrackingIntegration
{
      string TestConnection(string inputXML)
      { //implementation}

      string SubmitDefectToTool(string inputXML)
      { //implementation}

      string GetDefectFieldListFromTool(string inputXML)
      {//implementation}

      string GetDefectListFromTool(string inputXML)
      { //implementation}

      string LaunchDefectTool(string inputXML)
      { //implementation}

      string EditDefectItem(string inputXML)
      { //implementation}
}
```

Parameter Information

The parameters (`string inputXML`) send the integration parameters and their values as defined in the tool and tool domain. Depending on which method is called, the inputXML may contain other information such as fields to include when submitting a defect.

Return Value Information

- In order for *QA*Director to process the returned data, the return XML string *must* contain the values or error message produced by your implementation of the integration.
- All element names in the return XML must be exactly as the appear in the examples in this section. For example, the element `returncode` cannot be `ReturnCode`.
- If the element `returncode` has `value="0"`, use the `message` element to set an error message.

Member Information

Refer to the following for examples of each member of the class:

- EditDefectItem [p. 35]
- GetDefectFieldListFromTool [p. 22]
- GetDefectListFromTool [p. 26]
- LaunchDefectTool [p. 34]
- SubmitDefectToTool [p. 17]
- TestConnection [p. 15]

TestConnection

Tests the connection to the defect tool by using the integration parameters provided in the defect Tool Domain.

Syntax

`TestConnection(string inputXML)`

Input XML

The input string is an XML string that is automatically generated by *QA*Director based on the tool and tool domain settings:

```
<root>
   <properties category="defect" id="" name="">
      <property name="edittool" type="text" value="JiraDefectIntegration.dll" />
      <property name="submittool" type="text" value="JiraDefectIntegration.dll" />
      <property name="retrievetool" type="text" value="JiraDefectIntegration.dll" />
      <property name="Database Server" type="text" value="dtwlib4m-073" />
      <property name="Port Number" type="text" value="8090" />
      <property name="User Name" type="text" value="admin" />
      <property name="User Password" type="text" value="admin" />
      <property name="Project Key" type="text" value="" />
   </properties>
 </root>
```

Return Value

Returns output in an XML string. Returncode value of 1 means success and 0 means error occurred.

```
<root>
  <results>
    <result name="message" value=""/>
    <result name="returncode" value="1"/>
  </results>
</root>
```

Code Sample

```
public string TestConnection(string inputXML)
{
   int Step = 0;
   string User = "";
   string Password = "";
   string DBServer = "";
   string Port = "";
   string token = "";
   XmlDocument rtnXmlDoc = new XmlDocument();
   string XmlResultContent = "";
   string RS = "";
   Jira.JiraSoapServiceService soapService = new Jira.JiraSoapServiceService();

   try
   {
     //-------------------------------------------------------
     //Retrieve Tool Domain parameters from QADirector
     //-------------------------------------------------------
     XmlDocument xmlDoc = new XmlDocument();
     xmlDoc.LoadXml(inputXML);
     string path = "/root/properties/property";
     XmlNodeList Nodes = xmlDoc.SelectNodes(path);

     for (int i = 0; i < Nodes.Count; i++)
     {
       switch (Nodes[i].Attributes["name"].Value)
       {
          case "User Name": User = Nodes[i].Attributes["value"].Value; break;
          case "User Password": Password = Nodes[i].Attributes["value"].Value; break;
          case "Server Name": DBServer = Nodes[i].Attributes["value"].Value; break;
          case "Port Number": Port = Nodes[i].Attributes["value"].Value; break;
        }
     }
```

```
     Step = 1;

     //---------------------------------------------------------
     //Test Login
     //---------------------------------------------------------
     soapService.Url = "http://" + DBServer + ":" + Port +
"/rpc/soap/jirasoapservice-v2?wsdl";

     token = "";
     try
     {
       token = soapService.login(User, Password);
     }
     catch (Exception e)
     {
       XmlResultContent = "<result name='message' value='" + CleanMsg(e.Message) + "'
 /><result name='returncode' value='" + ErrorCode + "' />";
       RS = XMLBegin + XMLMiddle + XmlResultContent + XMLEnd;
       return RS;
     }

     RS = XMLBegin + XMLMiddle + XMLEnd;
   }
   catch (Exception ex)
   {
     if (Step == 1)
     {
       XmlResultContent = "<result name='message' value='Could not login to the Jira
instance specified. Check the Jira Product Integration settings. Exception: " +
CleanMsg(ex.Message) + "' /><result name='returncode' value='" + ErrorCode + "' />";
     }
     else
     {
       XmlResultContent = "<result name='message' value='" + CleanMsg(ex.Message) + "'
 /><result name='returncode' value='" + ErrorCode + "' />";
       soapService.logout(token);
     }

     RS = XMLBegin + XMLMiddle + XmlResultContent + XMLEnd;
     }

     finally
     {
       if (token != null)
       {
         try
         {
           soapService.logout(token);
         }
         catch
         {}
       }
     }
   return RS;
 }
```

SubmitDefectToTool

Submits a defect to the defect tool.

Syntax

`SubmitDefectToTool(string inputXML)`

Parameters

The input string is an XML string that is auto generated by *QA*Director based on the tool and tool domain settings. The tool domain integration parameters and submit defect field values for the selected fields are passed into the input xml.

```
<root>
    <properties category="defect" id="" name="">
       <property name="toolcomponent" type="text" value="JiraDefectIntegration.dll" />

        <property name="projectid" type="text" value="108" />
        <property name="ssoname" type="text" value="admin" />
        <property name="ssopassword" type="text" value="admin" />
        <property name="Database Server" type="text" value="dtwlib4m-073" />
        <property name="Port Number" type="text" value="8090" />
        <property name="User Name" type="text" value="admin" />
        <property name="User Password" type="text" value="admin" />
        <property name="Project Key" type="text" value="TESTING" />
    </properties>
    <fields category="defect" id="" name="">
       <field category="text" name="PropagatedRequirements" datatype="text" value="New
 Requirement" selected="1" fieldtype="" />
        <field category="text" name="DefectSummary" datatype="text" value="Jira Test"
selected="1" fieldtype="" />
      <field category="text" name="DefectDescription" datatype="text" value="Description
 of the TEST" selected="1" fieldtype="" />
       <field category="text" name="JobName" datatype="text" value="Jira Test Job Name"
 selected="1" fieldtype="" />
        <field category="text" name="JobId" datatype="text" value="5" selected="1"
fieldtype="" />
        <field category="text" name="AssetId" datatype="text" value="117" selected="1"
 fieldtype="" />
        <field category="text" name="AssetName" datatype="text" value="Jira Test"
selected="1" fieldtype="" />
      <field category="text" name="AssetType" datatype="text" value="Test" selected="1"
 fieldtype="" />
      <field category="text" name="AssetDescription" datatype="text" value="Description
 of the TEST" selected="1" fieldtype="" />
      <field category="text" name="InstanceId" datatype="text" value="16" selected="1"
 fieldtype="" />
       <field category="text" name="FailureDescription" datatype="text" value="this is
 a failure description" selected="1" fieldtype="" />
        <field category="text" name="TestingToolName" datatype="text" value=""
selected="1" fieldtype="" />
      <field category="text" name="ToolDomainName" datatype="text" value="" selected="1"
 fieldtype="" />
      <field category="text" name="TestExecutedBy" datatype="text" value="" selected="1"
 fieldtype="" />
      <field category="text" name="TestStartTime" datatype="text" value="" selected="1"
 fieldtype="" />
       <field category="text" name="TestEndTime" datatype="text" value="" selected="1"
 fieldtype="" />
        <field category="text" name="ExecutionPlanName" datatype="text"
value="EP.PlanName..Jira" selected="1" fieldtype="" />
        <field category="text" name="ExecutionMachineName" datatype="text" value=""
selected="1" fieldtype="" />
        <field category="text" name="ExecutionMachineOS" datatype="text" value=""
selected="1" fieldtype="" />
    </fields>
</root>
```

Return Value

Returns output in an XML string. Returncode value of 1 means success and 0 means error occurred.

```
<root>
    <results>
       <result name="message" value="" />
       <result name="returncode" value="1" />
       <result name="defectid" value="10052" />
       <result name="displayid" value="TESTING-45" />
```

```
    </results>
</root>
```

Code Sample

```
public string SubmitDefectToTool(string inputXML)
{
  #region variable declarations
  int Step = 0;
  string RS = "";
  string User = "";
  string Password = "";
            string DBServer = "";
  string Port = "";
  string ProjectKey = "";
  string token = "";
  string SSOUser = "";
  string SSOPassword = "";
  string XmlContent = "";
  string strCommon = "-----QAD Specific Fields-----\n";
  string strCommonExec = "---Execution Details---\n";
  string strDescription = "";
  string strPropogatedTR = "";
  string strTestDesc = "";
  string strTestOwnerName = "";
  string strJobName = "";
  string strScriptName = "";
  string strScriptDesc = "";
  string strFailureDesc = "";
  string strStatus = "";
  string strDefectSummary = "";
  string strStartTime = "";
  string strEndTime = "";
  string strExecMach = "";
#endregion

  Jira.JiraSoapServiceService soapService = new Jira.JiraSoapServiceService();
  Jira.RemoteIssue issue = new JiraDefectIntegration.Jira.RemoteIssue();
  Jira.RemoteComment comment = new JiraDefectIntegration.Jira.RemoteComment();

  try
  {
      XmlDocument xmlDoc = new XmlDocument();
      xmlDoc.LoadXml(inputXML);
      string path = "/root/properties/property";
      XmlNodeList Nodes = xmlDoc.SelectNodes(path);

      //--------------------------------------------------------
      //Retrieve Tool Domain parameters from QADirector
      //--------------------------------------------------------
      for (int i = 0; i < Nodes.Count; i++)
      {

      switch (Nodes[i].Attributes["name"].Value)
      {
       case "User Name":
           User = Nodes[i].Attributes["value"].Value; break;

       case "User Password":
           Password = Nodes[i].Attributes["value"].Value; break;
       case "Server Name":
           DBServer = Nodes[i].Attributes["value"].Value; break;
       case "Port Number":
           Port = Nodes[i].Attributes["value"].Value; break;
       case "Project Key":
           ProjectKey = Nodes[i].Attributes["value"].Value; break;
       case "ssoname":
           SSOUser = Nodes[i].Attributes["value"].Value; break;
       case "ssopassword":
           SSOPassword = Nodes[i].Attributes["value"].Value; break;
       }
      }

      path = "/root/fields/field";
```

```
     Nodes = xmlDoc.SelectNodes(path);

     //-------------------------------------------------------
     //Retrieve information to Submit Issue
     //-------------------------------------------------------
     for (int i = 0; i < Nodes.Count; i++)
     {
  if (Nodes[i].Attributes["category"].Value != "file")
  {
   switch (EscapeQuotes(Nodes[i].Attributes["name"].Value))
   {
       case SubmitDefectFields.DefectSynopsis:
          strDefectSummary = EscapeQuotes(Nodes[i].Attributes["value"].Value); break;

       case SubmitDefectFields.Status:
            strStatus = EscapeQuotes(Nodes[i].Attributes["value"].Value);
            issue.status = strStatus;  break;
       case SubmitDefectFields.DefectDescription:
          strDescription = EscapeQuotes(Nodes[i].Attributes["value"].Value) + "\n\n";
break;
       case SubmitDefectFields.PropagatedTRs:
            strPropogatedTR += EscapeQuotes(Nodes[i].Attributes["value"].Value);
            if (strPropogatedTR == string.Empty)
            { strPropogatedTR = "  -  "; }
            strCommon += "Test Requirments: " + strPropogatedTR + "\n";  break;
       case SubmitDefectFields.TestDesc:
            strTestDesc = EscapeQuotes(Nodes[i].Attributes["value"].Value);
            if (strTestDesc == string.Empty)
            { strTestDesc = "  -  "; }
            strCommon += "Test Summary: " + strTestDesc + "\n"; break;
       case SubmitDefectFields.TestOwnerName:
            strTestOwnerName = EscapeQuotes(Nodes[i].Attributes["value"].Value);
            strCommon += "Test Owner Name: " + strTestOwnerName + "\n"; break;
       case SubmitDefectFields.JobName:
            strJobName = EscapeQuotes(Nodes[i].Attributes["value"].Value);
            if (strJobName == string.Empty)
            { strJobName = "  -  "; }
            strCommon += "Job Name: " + strJobName + "\n";
       case SubmitDefectFields.ScriptName:
            strScriptName = EscapeQuotes(Nodes[i].Attributes["value"].Value);
            if (strScriptName == string.Empty)
            { strScriptName = "  -  "; }
            strCommon += "Script Name: " + strScriptName + "\n"; break;
       case SubmitDefectFields.ScriptDesc:
            strScriptDesc = EscapeQuotes(Nodes[i].Attributes["value"].Value);
            if (strScriptDesc == string.Empty)
            { strScriptDesc = "  -  "; }
            strCommon += "Script Description: " + strScriptDesc + "\n"; break;
       case SubmitDefectFields.FailureDesc:
            strFailureDesc = EscapeQuotes(Nodes[i].Attributes["value"].Value);
            if (strFailureDesc == string.Empty)
            { strFailureDesc = "  -  "; }
            strCommon += "Failure Description: " + strFailureDesc + "\n"; break;
        case SubmitDefectFields.ExecMachine:
            strExecMach = EscapeQuotes(Nodes[i].Attributes["value"].Value);
            if (strExecMach == string.Empty)
            { strExecMach = "  -  "; }
            issue.environment = "Execution Machine: " + strExecMach + "\n";
            strCommonExec += "Execution Machine Name: " + strExecMach + "\n";  break;
       case SubmitDefectFields.StartTime:
            strStartTime = EscapeQuotes(Nodes[i].Attributes["value"].Value);
            strCommonExec += "Test Start: " + strStartTime + "\n"; break;
       case SubmitDefectFields.EndTime:
            strEndTime = EscapeQuotes(Nodes[i].Attributes["value"].Value.ToString());
            strCommonExec += "Test End: " + strEndTime + "\n";  break;
    }
   }
  }

     issue.type = "1"; // Makes the issue type a "Bug"
     issue.description = strDescription + strCommon + strCommonExec;
     issue.summary = strDefectSummary;

     Step = 1;
```

```
      //-------------------------------------------------------
      //Login
      //-------------------------------------------------------
      soapService.Url = "http://" + DBServer + ":" + Port +
"/rpc/soap/jirasoapservice-v2?wsdl";
      token = "";
      try
      {
         token = soapService.login(SSOUser, SSOPassword);
      }
      catch (Exception e)
      {
         XmlContent = "<result name='message' value='" + CleanMsg(e.Message) + "'
/><result name='returncode' value='" + ErrorCode + "' />";
         RS = XMLBegin + XMLMiddle + XmlContent + XMLEnd;
         return RS;
      }

      Step = 2;

      //-------------------------------------------------------
      //Retrieve Projects & Submit Issue
      //-------------------------------------------------------
      try
      {
        issue.project = ProjectKey;

        //find the project
        Jira.RemoteProject project = new JiraDefectIntegration.Jira.RemoteProject();
        project = soapService.getProjectByKey(token, ProjectKey);
        if (project != null)
        {   //if the project is found, assign this issue to the team lead (which is
default in JIRA)
           issue.assignee = project.lead;
        }
        else
        {   //if the project is not found, assign this issue to the current
            // user so that the issue is created with out any problem
            issue.assignee = SSOUser;
         }

      Jira.RemoteIssue createdIssue;
      createdIssue = soapService.createIssue(token, issue);
     XmlContent = "<result name='message' value='' /><result name='returncode' value='"
 + SuccessCode + "' /><result name='defectid' value='" + createdIssue.id + "' /><result
 name='displayid' value='" + createdIssue.key + "' />";
      }
      catch (Exception ex)
      {
          XmlContent = "<result name='message' value='Defect submission failure. " +
CleanMsg(ex.Message) + "' /><result name='returncode' value='" + ErrorCode + "' /><result
 name='defectid' value='" + RS + "' />";
      }
     }
     catch (Exception ex)
     {
      if (Step == 1)
       XmlContent = "<result name='message' value='Could not login to the Jira instance
 specified. Check the Product Integration settings. Exception:" + CleanMsg(ex.Message)
 + "' /><result name='returncode' value='" + ErrorCode + "' /><result name='defectid'
value='" + RS + "' />";
      else
      {
     if (Step == 2)
     {
        XmlContent = "<result name='message' value='Unable to submit item.  Required
Fields not set correctly for Tool Domain. Exception:" + CleanMsg(ex.Message) + "'
/><result name='returncode' value='" + ErrorCode + "' /><result name='defectid' value='"
 + RS + "' />";
     }
     else
        XmlContent = "<result name='message' value='" + CleanMsg(ex.Message) + "'
/><result name='returncode' value='" + ErrorCode + "' /><result name='defectid' value='"
 + RS + "' /><result name='displayid' value='" + RS + "' />";
      }
```

```
    }
   finally
   {
     soapService.logout(token);
    }

    RS = XMLBegin + XMLMiddle + XmlContent + XMLEnd;
    return RS;
}
```

GetDefectFieldListFromTool

Retrieves the fields information of the defect item in the defect tool.

Syntax

`GetDefectFieldListFromTool(string inputXML)`

Parameters

`inputXML` is an xml string.

```
<root>
    <properties category="defect" id="" name="">
    <property name="toolcomponent" type="text" value="JiraDefectIntegration.dll" />
    <property name="Database Server" type="text" value="dtwlib4m-073" />
    <property name="Port Number" type="text" value="8090" />
    <property name="User Name" type="text" value="admin" />
    <property name="User Password" type="text" value="admin" />
    <property name="Project Key" type="text" value="" />
    </properties>
</root>
```

Return Value

Returns a list of defect fields in an XML string. Returncode value of 1 means success and 0 means an error occurred.

```
<root>
    <fields>
        <field name="key" datatype="TEXT" selected="1" reqfieldtype="DISPLAYID" />
        <field name="id" datatype="NUMERIC" selected="1" reqfieldtype="UNIQUEID" />
        <field name="Type" datatype="TEXT" selected="0" reqfieldtype="OTHER"
caption="Type" />
        <field name="status" datatype="TEXT" selected="1" reqfieldtype="STATUS" />
        <field name="resolution" datatype="NUMERIC" selected="0" reqfieldtype="OTHER"
caption="Resolution" />
        <field name="priority" datatype="TEXT" selected="1" reqfieldtype="PRIORITY" />

        <field name="Assignee" datatype="TEXT" selected="0" reqfieldtype="OTHER"
caption="Assignee" />
        <field name="Reporter" datatype="TEXT" selected="0" reqfieldtype="OTHER"
caption="Reporter" />
        <field name="summary" datatype="TEXT" selected="1" reqfieldtype="SUMMARY" />
        <field name="Enviroment" datatype="TEXT" selected="0" reqfieldtype="OTHER"
caption="Enviroment" />
        <field name="Description" datatype="TEXT" selected="0" reqfieldtype="OTHER"
caption="Description" />
        <field name="Comments" datatype="TEXT" selected="0" reqfieldtype="OTHER"
caption="Comments" />
        <field name="DueDate" datatype="TEXT" selected="0" reqfieldtype="OTHER"
caption="DueDate" />
       <field name="datecreated" datatype="DATE" selected="1" reqfieldtype="DATECREATED"
 />
```

```
    </fields>
</root>
```

Example

```
public string GetDefectFieldListFromTool(string inputXML)
        {
            string SelectedYes = "1";
            string SelectedNo = "0";
            int step = 0;
            string User = "";
            string Password = "";
            string DBServer = "";
            string Port = "";
            string ProjectKey = "";
            string token = "";
            string SSOUser = "";
            string SSOPassword = "";
            string XmlResultContent = "";
            XmlDocument rtnXmlDoc = new XmlDocument();
            string RS = "";

           Jira.JiraSoapServiceService soapService = new Jira.JiraSoapServiceService();


            try
            {
                XmlDocument xmlDoc = new XmlDocument();
                xmlDoc.LoadXml(inputXML);
                string path = "/root/properties/property";
                XmlNodeList Nodes = xmlDoc.SelectNodes(path);

                //---------------------------------------------------------
                //Retrieve Tool Domain parameters from QADirector
                //---------------------------------------------------------
                for (int i = 0; i < Nodes.Count; i++)
                {
                    switch (Nodes[i].Attributes["name"].Value)
                    {

                        case "User Name":
                            User = Nodes[i].Attributes["value"].Value;
                            break;
                        case "User Password":
                            Password = Nodes[i].Attributes["value"].Value;
                            break;
                        case "Server Name":
                            DBServer = Nodes[i].Attributes["value"].Value;
                            break;
                        case "Port Number":
                            Port = Nodes[i].Attributes["value"].Value;
                            break;
                        case "Project Key":
                            ProjectKey = Nodes[i].Attributes["value"].Value;
                            break;
                        case "ssoname":
                            SSOUser = Nodes[i].Attributes["value"].Value;
                            break;
                        case "ssopassword":
                            SSOPassword = Nodes[i].Attributes["value"].Value;
                            break;
                    }
                }

                XmlElement RootElem = rtnXmlDoc.CreateElement("root");
                rtnXmlDoc.AppendChild(RootElem);

                step = 1;

                //---------------------------------------------------------
                //Test Login
                //---------------------------------------------------------
                soapService.Url = "http://" + DBServer + ":" + Port +
"/rpc/soap/jirasoapservice-v2?wsdl";
```

```
                token = "";
                try
                {
                    token = soapService.login(User, Password);
                }
                catch (Exception e)
                {
                    XmlResultContent = "<result name='message' value='" +
CleanMsg(e.Message) + "' /><result name='returncode' value='" + ErrorCode + "' />";
                    RS = XMLBegin + XMLMiddle + XmlResultContent + XMLEnd;
                    return RS;
                }

                step = 2;

                //-------------------------------------------------------
                //Gather Tool Domain Data
                //-------------------------------------------------------
                XmlElement FieldsElem = rtnXmlDoc.CreateElement("fields");
                RootElem.AppendChild(FieldsElem);

                StringCollection tmpFields = new StringCollection();
                tmpFields.AddRange(getFieldNames());

                for (int j = 0; j < tmpFields.Count; j++)
                {
                    string field = tmpFields[j];

                    XmlElement FieldElem = this.CreateFieldNode(rtnXmlDoc,
EncodeXmlValue(field), "", SelectedNo, "");

                    if (field.CompareTo(JiraFields.KEY) == 0)
                    {
                        FieldElem.SetAttribute(XMLConstants.FIELD_DATATYPE_ATTR,
FieldDataTypes.Text);
                        FieldElem.SetAttribute(XMLConstants.FIELD_REQFIELDTYPE_ATTR,
RequiredFieldTypes.DisplayID);
                        FieldElem.SetAttribute(XMLConstants.FIELD_SELECTED_ATTR,
SelectedYes);
                    }
                    else if (field.CompareTo(JiraFields.ID) == 0)
                    {
                        FieldElem.SetAttribute(XMLConstants.FIELD_DATATYPE_ATTR,
FieldDataTypes.Numeric);
                        FieldElem.SetAttribute(XMLConstants.FIELD_REQFIELDTYPE_ATTR,
RequiredFieldTypes.UniqueID);
                        FieldElem.SetAttribute(XMLConstants.FIELD_SELECTED_ATTR,
SelectedYes);
                    }
                    else if (field.CompareTo(JiraFields.SUMMARY) == 0)
                    {
                        FieldElem.SetAttribute(XMLConstants.FIELD_DATATYPE_ATTR,
FieldDataTypes.Text);
                        FieldElem.SetAttribute(XMLConstants.FIELD_REQFIELDTYPE_ATTR,
RequiredFieldTypes.Summary);
                        FieldElem.SetAttribute(XMLConstants.FIELD_SELECTED_ATTR,
SelectedYes);
                    }
                    else if (field.CompareTo(JiraFields.STATUS) == 0)
                    {
                        FieldElem.SetAttribute(XMLConstants.FIELD_DATATYPE_ATTR,
FieldDataTypes.Text);
                        FieldElem.SetAttribute(XMLConstants.FIELD_REQFIELDTYPE_ATTR,
RequiredFieldTypes.Status);
                        FieldElem.SetAttribute(XMLConstants.FIELD_SELECTED_ATTR,
SelectedYes);
                    }
                    else if (field.CompareTo(JiraFields.PRIORITY) == 0)
                    {
                        FieldElem.SetAttribute(XMLConstants.FIELD_DATATYPE_ATTR,
FieldDataTypes.Text);
                        FieldElem.SetAttribute(XMLConstants.FIELD_REQFIELDTYPE_ATTR,
RequiredFieldTypes.Priority);
                        FieldElem.SetAttribute(XMLConstants.FIELD_SELECTED_ATTR,
SelectedYes);
```

```
                    }
                    else if (field.CompareTo(JiraFields.RESOLUTION) == 0)
                    {
                        FieldElem.SetAttribute(XMLConstants.FIELD_DATATYPE_ATTR,
FieldDataTypes.Numeric);
                        FieldElem.SetAttribute(XMLConstants.FIELD_REQFIELDTYPE_ATTR,
RequiredFieldTypes.Other);
                        FieldElem.SetAttribute(XMLConstants.FIELD_NAME_ATTR,
"resolution");
                        FieldElem.SetAttribute(XMLConstants.FIELD_CAPTION_ATTR,
"Resolution");
                        FieldElem.SetAttribute(XMLConstants.FIELD_SELECTED_ATTR,
SelectedNo);
                    }
                    else if (field.CompareTo(JiraFields.DATE_CREATED) == 0)
                    {
                        FieldElem.SetAttribute(XMLConstants.FIELD_DATATYPE_ATTR,
FieldDataTypes.Date);
                        FieldElem.SetAttribute(XMLConstants.FIELD_REQFIELDTYPE_ATTR,
RequiredFieldTypes.DateCreated);
                        FieldElem.SetAttribute(XMLConstants.FIELD_SELECTED_ATTR,
SelectedYes);
                    }
                    else if (field.CompareTo(JiraFields.DUE_DATE) == 0)
                    {
                        FieldElem.SetAttribute(XMLConstants.FIELD_DATATYPE_ATTR,
FieldDataTypes.Text);
                        FieldElem.SetAttribute(XMLConstants.FIELD_REQFIELDTYPE_ATTR,
RequiredFieldTypes.Other);
                       FieldElem.SetAttribute(XMLConstants.FIELD_NAME_ATTR, "DueDate");

                        FieldElem.SetAttribute(XMLConstants.FIELD_CAPTION_ATTR,
"DueDate");
                        FieldElem.SetAttribute(XMLConstants.FIELD_SELECTED_ATTR,
SelectedNo);
                    }
                    else
                    {
                        FieldElem.SetAttribute(XMLConstants.FIELD_DATATYPE_ATTR,
FieldDataTypes.Text);
                        FieldElem.SetAttribute(XMLConstants.FIELD_NAME_ATTR, field);
                       FieldElem.SetAttribute(XMLConstants.FIELD_CAPTION_ATTR, field);

                        FieldElem.SetAttribute(XMLConstants.FIELD_REQFIELDTYPE_ATTR,
RequiredFieldTypes.Other);
                    }
                    FieldsElem.AppendChild(FieldElem);
                }

                if (FieldsElem == null)
                {
                    XmlResultContent = "<result name='message' value='Could not find
the specified Jira Product in the  database.' /><result name='returncode' value='" +
ErrorCode + "' />";
                    RS = XMLBegin + XMLMiddle + XmlResultContent + XMLEnd;

                    if (soapService.login(User, Password) == null)
                    {
                        soapService.logout(token);
                    }

                    return RS;
                }

                RS = rtnXmlDoc.InnerXml;

            }
            catch (Exception ex)
            {
                if (step == 1)
                {
                    XmlResultContent = "<result name='message' value='Could not login
to the Jira instance specified. Check the Product Integration settings. Exception: " +
 CleanMsg(ex.Message) + "' /><result name='returncode' value='" + ErrorCode + "' />";
                }
```

```
                else
                {
                    XmlResultContent = "<result name='message' value='" +
CleanMsg(ex.Message) + "' /><result name='returncode' value='" + ErrorCode + "' />";
                }

                RS = XMLBegin + XMLMiddle + XmlResultContent + XMLEnd;
            }
            finally
            {
                try
                {
                    soapService.logout(token);
                }
                catch
                {}

            }

            return RS;
        }
```

GetDefectListFromTool

Retrieves the defect data from the defect tool.

Syntax

`GetDefectListFromTool(string inputXML)`

Parameters

`string inputXML`

```
<root>
    <fields>
        <field name="key" datatype="TEXT" selected="1" reqfieldtype="DISPLAYID" />
        <field name="id" datatype="NUMERIC" selected="1" reqfieldtype="UNIQUEID" />
        <field name="Type" datatype="TEXT" selected="0" reqfieldtype="OTHER"
caption="Type" />
        <field name="status" datatype="TEXT" selected="1" reqfieldtype="STATUS" />
        <field name="resolution" datatype="NUMERIC" selected="0" reqfieldtype="OTHER"
caption="Resolution" />
        <field name="priority" datatype="TEXT" selected="1" reqfieldtype="PRIORITY" />

        <field name="Assignee" datatype="TEXT" selected="0" reqfieldtype="OTHER"
caption="Assignee" />
        <field name="Reporter" datatype="TEXT" selected="0" reqfieldtype="OTHER"
caption="Reporter" />
        <field name="summary" datatype="TEXT" selected="1" reqfieldtype="SUMMARY" />
        <field name="Enviroment" datatype="TEXT" selected="0" reqfieldtype="OTHER"
caption="Enviroment" />
        <field name="Description" datatype="TEXT" selected="0" reqfieldtype="OTHER"
caption="Description" />
        <field name="Comments" datatype="TEXT" selected="0" reqfieldtype="OTHER"
caption="Comments" />
        <field name="DueDate" datatype="TEXT" selected="0" reqfieldtype="OTHER"
caption="DueDate" />
       <field name="datecreated" datatype="DATE" selected="1" reqfieldtype="DATECREATED"
 />
      </fields>
</root>
```

Return Value

Returns an XML string with defects and field information. Returncode value of 1 means success and 0 means error occurred. The following is an example, see **XML Schema Return Format** below for the full schema.

```
<root>
   <items>
      <item id="10051">
         <fields>
            <field name="key" value="TESTING-44" />
            <field name="id" value="10051" />
            <field name="status" value="Open" />
            <field name="priority" value="" />
            <field name="summary" value="Jira Test" />
            <field name="datecreated" value="7/29/2008" />
         </fields>
      </item>
      <item id="10050">
         <fields>
            <field name="key" value="TESTING-43" />
            <field name="id" value="10050" />
            <field name="status" value="Open" />
            <field name="priority" value="" />
            <field name="summary" value="Jira Test" />
            <field name="datecreated" value="7/29/2008" />
         </fields>
      </item>
      <item id="10049">
         <fields>
            <field name="key" value="TESTING-42" />
            <field name="id" value="10049" />
            <field name="status" value="Open" />
            <field name="priority" value="" />
            <field name="summary" value="Jira Test" />
            <field name="datecreated" value="7/29/2008" />
         </fields>
      </item>
   </items>
</root>
```

XML Schema Return Format

The output XML is required to adhere to the following schema:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="root">
      <xs:complexType>
         <xs:sequence>
            <!-- items section begin -->
            <xs:element name="items" minOccurs="0" maxOccurs="1">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="fields" minOccurs="0" maxOccurs="1">
                                 <xs:complexType>
                                    <xs:sequence>
                                       <xs:element name="field" minOccurs="0"
maxOccurs="unbounded">
                                          <xs:complexType>
                                             <xs:attribute name="category"
type="xs:string" use="optional"></xs:attribute>
                                             <xs:attribute name="name" type="xs:string"
 use="required"></xs:attribute>
                                          <xs:attribute name="datatype" use="optional">

                                             <xs:simpleType>
                                                <xs:restriction base="xs:string">
                                                   <xs:enumeration
value="TEXT"></xs:enumeration>
                                                   <xs:enumeration
value="NUMERIC"></xs:enumeration>
                                                   <xs:enumeration
value="DATE"></xs:enumeration>
                                                </xs:restriction>
                                             </xs:simpleType>
                                          </xs:attribute>
```

```
                                          <xs:attribute name="value" type="xs:string"
 use="optional"></xs:attribute>
                                       <xs:attribute name="selected" use="optional">

                                             <xs:simpleType>
                                                <xs:restriction base="xs:string">
                                                   <xs:enumeration
value="0"></xs:enumeration>
                                                   <xs:enumeration
value="1"></xs:enumeration>
                                                </xs:restriction>
                                             </xs:simpleType>
                                          </xs:attribute>
                                          <xs:attribute name="reqfieldtype"
use="optional">
                                             <xs:simpleType>
                                                <xs:restriction base="xs:string">
                                                   <xs:enumeration
value="UNIQUEID"></xs:enumeration>
                                                   <xs:enumeration
value="DISPLAYID"></xs:enumeration>
                                                   <xs:enumeration
value="STATUS"></xs:enumeration>
                                                   <xs:enumeration
value="PRIORITY"></xs:enumeration>
                                                   <xs:enumeration
value="SUMMARY"></xs:enumeration>
                                                   <xs:enumeration
value="DATECREATED"></xs:enumeration>
                                                   <xs:enumeration
value="OTHER"></xs:enumeration>
                                                </xs:restriction>
                                             </xs:simpleType>
                                          </xs:attribute>
                                       </xs:complexType>
                                    </xs:element>
                                 </xs:sequence>
                              </xs:complexType>
                           </xs:element>
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:string"
use="required"></xs:attribute>
                     </xs:complexType>
                  </xs:element>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <!-- items section complete -->
         <!-- Results section begin -->
         <xs:element name="results" minOccurs="0" maxOccurs="1">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="result" minOccurs="0" maxOccurs="unbounded">
                     <xs:complexType>
                        <xs:attribute name="name" use="required">
                           <xs:simpleType>
                              <xs:restriction base="xs:string">
                                 <xs:enumeration value="message"></xs:enumeration>
                                  <xs:enumeration value="returncode"></xs:enumeration>
                                 <xs:enumeration value="defectid"></xs:enumeration>
                              </xs:restriction>
                           </xs:simpleType>
                        </xs:attribute>
                        <xs:attribute name="value" type="xs:string"
use="required"></xs:attribute>
                     </xs:complexType>
                  </xs:element>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <!-- Results section complete-->
      </xs:sequence>
   </xs:complexType>
```

```
   </xs:element>
</xs:schema>
```

Code Sample

```
public string GetDefectListFromTool(string inputXML)
        {
            string RS = inputXML;
            int Step = 0;
            string User = "";
            string Password = "";
            string DBServer = "";
            string Port = "";
            string ProjectKey = "";
            string token = "";
            string XmlResultContent = "";
            XmlDocument rtnXmlDoc = new XmlDocument();
           Jira.JiraSoapServiceService soapService = new Jira.JiraSoapServiceService();

            bool bProjectFound = false;

            try
            {
                XmlDocument xmlDoc = new XmlDocument();
                xmlDoc.LoadXml(inputXML);
                string path = "/root/properties/property";
                XmlNodeList Nodes = xmlDoc.SelectNodes(path);

                //---------------------------------------------------------
                //Retrieve Tool Domain parametets from QADirector
                //---------------------------------------------------------
                for (int i = 0; i < Nodes.Count; i++)
                {
                    switch (Nodes[i].Attributes["name"].Value)
                    {
                        case "User Name":
                            User = Nodes[i].Attributes["value"].Value;
                            break;
                        case "User Password":
                            Password = Nodes[i].Attributes["value"].Value;
                            break;
                        case "Server Name":
                            DBServer = Nodes[i].Attributes["value"].Value;
                            break;
                        case "Port Number":
                            Port = Nodes[i].Attributes["value"].Value;
                            break;
                        case "Project Key":
                            ProjectKey = Nodes[i].Attributes["value"].Value;
                            break;
                    }
                }

                XmlElement RootElem = rtnXmlDoc.CreateElement("root");
                rtnXmlDoc.AppendChild(RootElem);

                Step = 1;

                //---------------------------------------------------------
                //Login
                //---------------------------------------------------------
                soapService.Url = "http://" + DBServer + ":" + Port +
"/rpc/soap/jirasoapservice-v2?wsdl"; ;

                token = "";
                try
                {
                    token = soapService.login(User, Password);
                }
                catch (Exception e)
                {
                    XmlResultContent = "<result name='message' value='" +
CleanMsg(e.Message) + "' /><result name='returncode' value='" + ErrorCode + "' />";
                    RS = XMLBegin + XMLMiddle + XmlResultContent + XMLEnd;
```

```
                    return RS;
                }

                Step = 2;

               Jira.RemoteIssue[] issues = new JiraDefectIntegration.Jira.RemoteIssue[]
 { };
                Jira.RemoteProject project = new
JiraDefectIntegration.Jira.RemoteProject();

                XmlElement ItemsElem = rtnXmlDoc.CreateElement("items");
                RootElem.AppendChild(ItemsElem);

                project = soapService.getProjectByKey(token, ProjectKey);

                if (project != null)
                {
                    bProjectFound = true;
                }

                if (bProjectFound)
                {
                    //get all of the issues for this project
                    try
                    {
                        issues = soapService.getIssuesFromTextSearchWithProject(token,
 new string[] { project.key }, "", 999999999);
                        foreach (Jira.RemoteIssue issue in issues)
                        {
                            // Spit out the XML

                            // First, the defect node
                            XmlElement ItemElem = this.CreateItemNode(rtnXmlDoc,
issue.id.ToString());
                            ItemsElem.AppendChild(ItemElem);

                            // Now, the field nodes
                            XmlElement FieldsElem = rtnXmlDoc.CreateElement("fields");
                            ItemElem.AppendChild(FieldsElem);

                            // Loop thru all possible fields of a bug
                            StringCollection fldNames = new StringCollection();
                            fldNames.AddRange(getFieldNames());
                            for (int j = 0; j < fldNames.Count; j++)
                            {
                                //Find this field name in the fields xml
                                XmlNode FieldNode =
xmlDoc.SelectSingleNode(XMLConstants.ROOT_NODE + "/" +
                                                        XMLConstants.FIELDS_NODE + "/" +
XMLConstants.FIELD_NODE +
                                                       "[@" + XMLConstants.FIELD_NAME_ATTR + "='"
 + fldNames[j] + "']");

                                if (FieldNode != null)
                                {
                                    if (fldNames[j].CompareTo("id") == 0)
                                    {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue(issue.id)));
                                    }
                                    else if (fldNames[j].CompareTo("summary") == 0)
                                    {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue(issue.summary)));
                                    }
                                    else if (fldNames[j].CompareTo("key") == 0)
                                    {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue(issue.key)));
                                    }
                                    else if (fldNames[j].CompareTo("Type") == 0)
                                    {
```

```
                                        #region Type Names
                                        if (issue.type.CompareTo("1") == 0)
                                        {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue("Bug")));
                                        }
                                        else if (issue.type.CompareTo("2") == 0)
                                        {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue("New Feature")));
                                        }
                                        else if (issue.type.CompareTo("3") == 0)
                                        {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue("Task")));
                                        }
                                        else if (issue.type.CompareTo("4") == 0)
                                        {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue("Improvment")));
                                        }
                                        #endregion
                                    }
                                   else if (fldNames[j].CompareTo("datecreated") == 0)

                                    {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue(issue.created.Value.ToShortDateString())));
                                    }
                                    else if (fldNames[j].CompareTo("status") == 0)
                                    {
                                        #region Status Names
                                        if (issue.status.Equals("1"))
                                        {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue("Open")));
                                        }
                                        else if (issue.status.Equals("3"))
                                        {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j], EncodeXmlValue("In
 Progress")));
                                        }
                                        else if (issue.status.Equals("4"))
                                        {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue("Reopened")));
                                        }
                                        else if (issue.status.Equals("5"))
                                        {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue("Resolved")));
                                        }
                                        else if (issue.status.Equals("6"))
                                        {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue("Closed")));
                                        }
                                        #endregion
                                    }
                                    else if (fldNames[j].CompareTo("Resolution") == 0)
                                    {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue(issue.resolution)));
                                    }
```

```
                                    else if (fldNames[j].CompareTo("priority") == 0)
                                    {
                                        #region Priority Names
                                        try
                                        {
                                            if (issue.priority.EndsWith("1"))
                                            {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue("Blocker")));
                                            }
                                            else if (issue.priority.EndsWith("2"))
                                            {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue("Critical")));
                                            }
                                            else if (issue.priority.EndsWith("3"))
                                            {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue("Major")));
                                            }
                                            else if (issue.priority.EndsWith("4"))
                                            {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue("Minor")));
                                            }
                                            else if (issue.priority.EndsWith("5"))
                                            {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue("Trivial")));
                                            }
                                        }
                                        catch
                                        {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j], EncodeXmlValue("
 ")));
                                        }
                                        #endregion
                                    }
                                    else if (fldNames[j].CompareTo("Assignee") == 0)
                                    {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue(issue.assignee)));
                                    }
                                    else if (fldNames[j].CompareTo("Reporter") == 0)
                                    {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue(issue.reporter)));
                                    }
                                    else if (fldNames[j].CompareTo("Enviroment") == 0)
                                    {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue(issue.environment)));
                                    }
                                   else if (fldNames[j].CompareTo("Description") == 0)

                                    {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue(issue.description)));
                                    }
                                    else if (fldNames[j].CompareTo("DueDate") == 0)
                                    {
                                        #region DueDate Specs
                                        if (issue.duedate.Equals(null))
                                        {
```

```
FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue("-")));
                                        }
                                        else
                                        {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue(issue.duedate.Value.ToShortDateString())));
                                        }
                                        #endregion
                                    }
                                    else if (fldNames[j].CompareTo("Comments") == 0)
                                    {
                                        Jira.RemoteComment[] commentarray =
soapService.getComments(token, issue.key);

                                        foreach (Jira.RemoteComment comment in
commentarray)
                                        {

FieldsElem.AppendChild(this.CreateItemFieldNode(rtnXmlDoc, fldNames[j],
EncodeXmlValue(comment.body + " || By: " + comment.author)));
                                        }
                                    }
                                }
                            }
                        }
                        catch
                        {
                           XmlResultContent = "<result name='message' value='Could not find
 any issues in the specified Jira product " + CleanMsg(ProjectKey) + "in the Jira
database.' /><result name='returncode' value='" + ErrorCode + "' />";
                            RS = XMLBegin + XMLMiddle + XmlResultContent + XMLEnd;
                        }
                    }
                    else
                    {
                        XmlResultContent = "<result name='message' value='Could not find
the specified Jira product \"" + CleanMsg(ProjectKey) + "\" in the Jira database.'
/><result name='returncode' value='" + ErrorCode + "' />";
                        RS = XMLBegin + XMLMiddle + XmlResultContent + XMLEnd;

                        try
                        {
                            soapService.logout(token);
                        }
                        catch
                        { }
                        return RS;
                    }
                    RS = rtnXmlDoc.InnerXml;
                }

                catch (Exception ex)
                {
                    if (Step == 1)
                        XmlResultContent = "<result name='message' value='Could not login
to the Jira instance specified. Check the Product Integration settings. Exception: " +
 CleanMsg(ex.Message) + "' /><result name='returncode' value='" + ErrorCode + "' /><result
 name='defectid' value='' />";
                    else
                    {
                        XmlResultContent = "<result name='message' value='" +
CleanMsg(ex.Message) + "' /><result name='returncode' value='" + ErrorCode + "' /><result
 name='defectid' value='' />";
                    }

                    RS = XMLBegin + XMLMiddle + XmlResultContent + XMLEnd;
                }
                finally
                {
                    try
                    {
                        soapService.logout(token);
```

```
                }
                catch
                {}
            }

            return RS;
        }
```

LaunchDefectTool

Launches the defect tool.

Syntax

`LaunchDefectTool(string inputXML)`

Parameters

The input string is an XML string that is auto generated by *QA*Director based on the tool and tool domain settings.

```
<root>
    <properties category="defect" id="10026" name="">
       <property name="toolcomponent" type="text" value="JiraDefectIntegration.dll" />

        <property name="defectid" type="text" value="TESTING-19" />
        <property name="projectid" type="text" value="108" />
        <property name="ssoname" type="text" value="admin" />
        <property name="ssopassword" type="text" value="admin" />
        <property name="Database Server" type="text" value="dtwlib4m-073" />
        <property name="Port Number" type="text" value="8090" />
        <property name="User Name" type="text" value="admin" />
        <property name="User Password" type="text" value="admin" />
        <property name="Project Key" type="text" value="TESTING" />
    </properties>
</root>
```

Return Value

Returns output in an xml string. Returncode value of 1 means success and 0 means error occurred.

```
<root>
             <results>
                 <result name="message" value="" />
                 <result name="returncode" value="1" />
             </results>
         </root>
```

Code Sample

```
public string LaunchDefectTool(string inputXML)
{
   string XmlResultContent = "";
   string rtnXmlStr = "";
   string strJiraURL = "";
   string DBServer = "";
   string Port = "";

   XmlDocument xmlDoc = new XmlDocument();
   xmlDoc.LoadXml(inputXML);
   string path = "/root/properties/property";
   XmlNodeList Nodes = xmlDoc.SelectNodes(path);

   for (int i = 0; i < Nodes.Count; i++)
   {
     switch (Nodes[i].Attributes["name"].Value)
     {
```

```
        case "Server Name":
          DBServer = Nodes[i].Attributes["value"].Value;
          break;
        case "Port Number":
          Port = Nodes[i].Attributes["value"].Value;
          break;
      }
    }

    try
    {
      strJiraURL = "http://" + DBServer + ":" + Port;

      if (strJiraURL != "http://:")
      {
        LaunchURL(strJiraURL);
      }

      XmlResultContent = "<result name='message' value='' /><result name='returncode'
value='" + SuccessCode + "' />";
    }
    catch (Exception excep)
    {
      XmlResultContent = "<result name='message' value='\n\nError occurred launching
Jira: \n" + excep.Message + "'  /><result name='returncode' value='" + ErrorCode + "'
/>";
    }

    rtnXmlStr = XMLBegin + XMLMiddle + XmlResultContent + XMLEnd;
    return rtnXmlStr;
 }
```

EditDefectItem

Opens the given defect in the defect tool.

Syntax

`EditDefectItem(string inputXML)`

Parameters

`inputXML` is an xml string that is auto generated by *QA*Director based on the Tool and Tool Domain settings and the requested defect identifier.

```
<root>
   <properties category="defect" id="10026" name="">
      <property name="ischangepoint" type="text" value="0" />
      <property name="toolcomponent" type="text" value="JiraDefectIntegration.dll" />
      <property name="defectid" type="text" value="TESTING-19" />
      <property name="projectid" type="text" value="108" />
     <property name="Site Id" type="text" value="4bc4c8dd-27a2-4b4b-b4c5-75c72df7c5b4"
 />
      <property name="ssoname" type="text" value="admin" />
      <property name="ssopassword" type="text" value="admin" />
      <property name="Database Server" type="text" value="dtwlib4m-073" />
      <property name="Port Number" type="text" value="8090" />
      <property name="User Name" type="text" value="admin" />
      <property name="User Password" type="text" value="admin" />
      <property name="Project Key" type="text" value="TESTING" />
   </properties>
</root>
```

Return Value

Returns an XML string. Returncode value of 1 means success and 0 means an error occurred.

```
<root>
   <results>
```

```
     <result name="message" value="" />
     <result name="returncode" value="1" />
   </results>
</root>
```

Code Sample

```
public string EditDefectItem(string inputXML)
        {
            string XmlResultContent = "";
            string rtnXmlStr = "";
            string User = "";
            string Password = "";
            string DBServer = "";
            string Port = "";
            string DefectID = "";
            string strJiraEditURL = "";
            string SSOUser = "";
            string SSOPassword = "";
            XmlDocument xmlDoc = new XmlDocument();
            xmlDoc.LoadXml(inputXML);
            string path = "/root/properties/property";
            XmlNodeList Nodes = xmlDoc.SelectNodes(path);

            for (int i = 0; i < Nodes.Count; i++)
            {
                switch (Nodes[i].Attributes["name"].Value)
                {
                    case "User Name":
                        User = Nodes[i].Attributes["value"].Value;
                        break;
                    case "User Password":
                        Password = Nodes[i].Attributes["value"].Value;
                        break;
                    case "Server Name":
                        DBServer = Nodes[i].Attributes["value"].Value;
                        break;
                    case "Port Number":
                        Port = Nodes[i].Attributes["value"].Value;
                        break;
                     case "ssoname":
                        SSOUser = Nodes[i].Attributes["value"].Value;
                        break;
                     case "ssopassword":
                        SSOPassword = Nodes[i].Attributes["value"].Value;
                        break;
                }
            }

            XmlDocument xmlDocs = new XmlDocument();
            xmlDocs.LoadXml(inputXML);
            string paths = "/root/properties";
            XmlNodeList Node = xmlDocs.SelectNodes(paths);

            for (int i = 0; i < Node.Count; i++)
            {
                switch (Node[i].Attributes["category"].Value)
                {
                    case "defect":
                        DefectID = Node[i].Attributes["id"].Value;
                        break;
                }
            }

            try
            {
               strJiraEditURL = "http://" + DBServer + ":" + Port +
"/secure/EditIssue!default.jspa?id=" + DefectID + "&os_username=" + SSOUser +
"&os_password=" + SSOPassword;

                if (DefectID != null && DBServer != null && Port != null)
                {
                    LaunchURL(strJiraEditURL);
                    XmlResultContent = "<result name='message' value='' /><result
```

```
name='returncode' value='" + SuccessCode + "' />";
                }
            }
            catch (Exception excep)
            {
                XmlResultContent = "<result name='message' value='\n\nError occurred
launching Jira: \n" + excep.Message + "'  /><result name='returncode' value='" + ErrorCode
 + "' />";
            }

            rtnXmlStr = XMLBegin + XMLMiddle + XmlResultContent + XMLEnd;
            return rtnXmlStr;
        }
```

Deployment

1. After coding the integration DLL, the output binaries should be copied to the following location on the *QA*Director web server machine: `\Micro Focus\QADirector\TMServices\ThirdPartyIntegrations`. Your DLL will be deployed in the following manner:

   Automated Tools

   Your DLL will be downloaded from the server to the client when needed.

   Defect Tools

   - For defect retrieval, the DLL is executed on the server.
   - For defect editing, the DLL is always downloaded and executed on the client on demand.
   - For defect submission, the default behavior is silent submission on the server. However, the **Tool Properties** dialog box provides an option to submit defects on the client. If this option is selected, then the DLL will be download and executed on the client.

2. Create a **Tool** and **Tool Domain** in *QA*Director to allow *QA*Director to send and receive the appropriate information.

   - When creating a **Tool**, be sure to select the appropriate type from the **Tool Type** list. For example: for defects, select the **`Defect`** type. For automated tools, select the **`Automated`** type.
   - The **Tool/Tool Domain** integration parameters are created and configured based on the needs of the tool/integration:

     Automated Tools

     Parameters can be set to apply at the tool domain or the tool properties. By applying a parameter at the tool properties, a value can be set that can be used in the tool. Additional Parameters can be added.

     Defect Tools

     Parameters for defect tool domains can be set to apply at the tool domain or the project level. By applying a parameter at the project level, duplicate tool domains can be avoided. The integration parameter values are set in two locations. Parameters that apply across the tool domain are set in the tool domain properties

integration parameters tab, while parameters that apply to projects are set in the project properties defect tracking tab.

- **Single Sign On**

  Automated Tools
  You can optionally use **Single Sign On** for the integration.

  Defect Tools
  Single Sign On is required for Defect submission and editing. They use the defect tool login specified in single sign on.

  For more information on Single Sign On and *QA*Director/third-party integrations, search for the following topic in the *QA*Director online help: *Integrating with External Products*.

- For Defect Tool integrations, be sure to associate the Project with the Tool Domain.

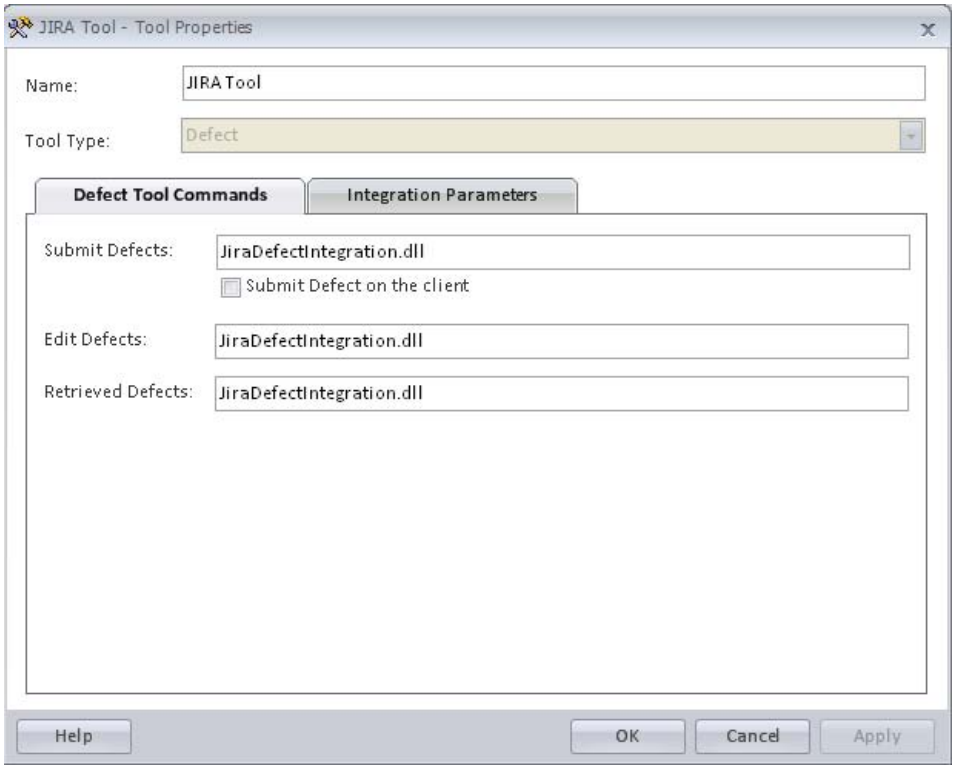
Setting Up a Defect Integration

This example walks you through how to set up a defect tool integration in the *QA*Director client application after your integration assemblies are compiled. The example below uses JIRA. Note that if you are integrating with JIRA, these steps are not necessary because a JIRATool and Tool Domain are already shipped. Refer to the *QA*Director online help for instructions on how to connect a JIRA database to a *QA*Director project.

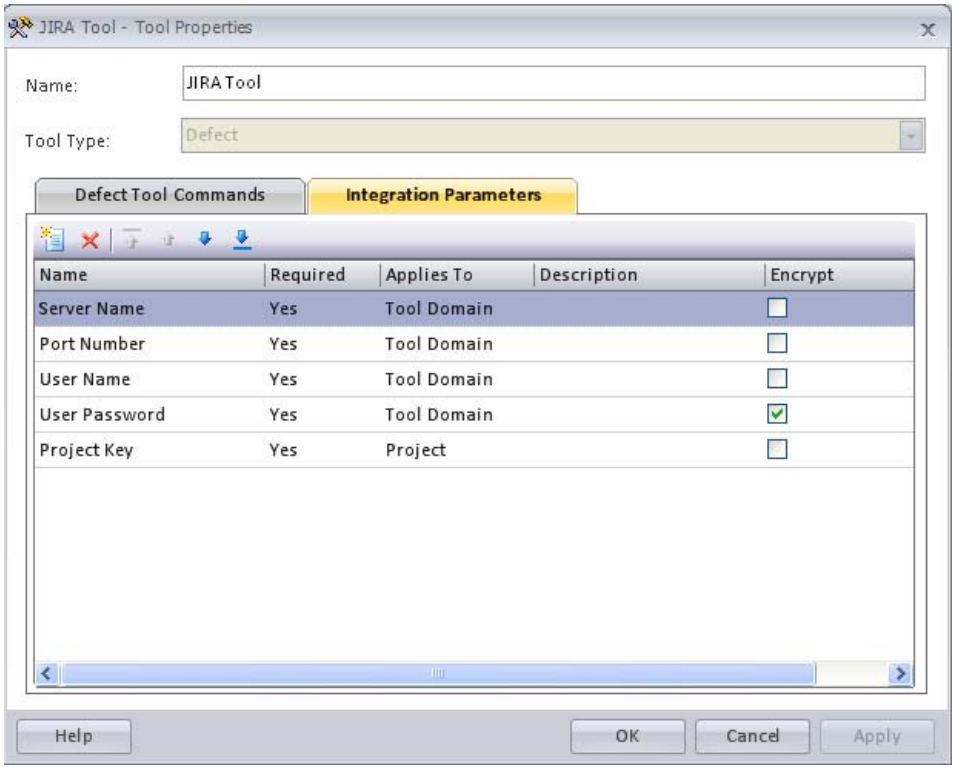
1. Copy your integration assemblies into the following destination directory:

```
\\Micro Focus\QADirector\TMServices\ThirdPartyIntegrations
```

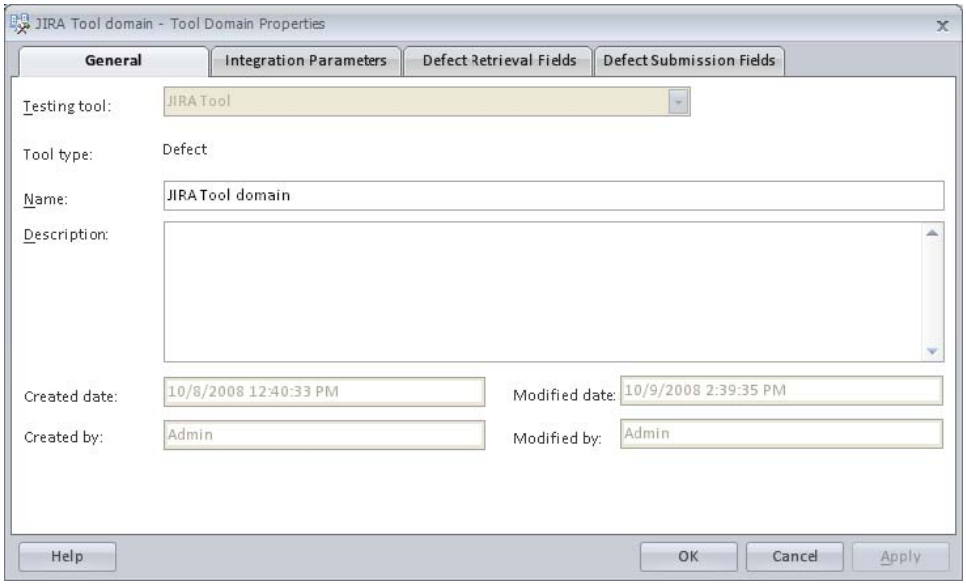
2. Create a JIRA tool by providing the appropriate defect tool commands (dll names) and Integration Parameters. Ensure that the Integration parameter names and settings are exactly as shown below. Select the **Defect Tool Commands** tab and enter the fields below.



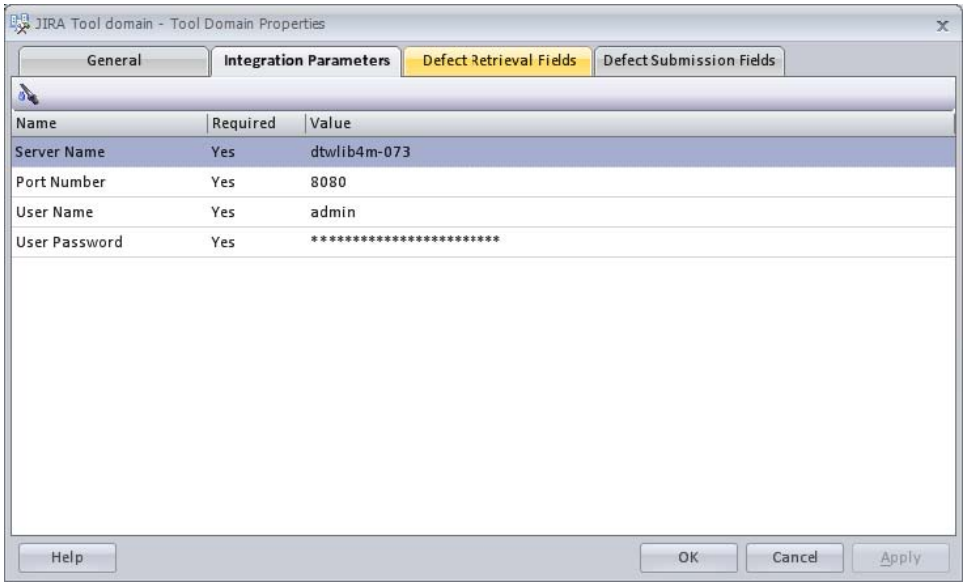
3. Select the **Integration Parameters** tab and enter the fields below:



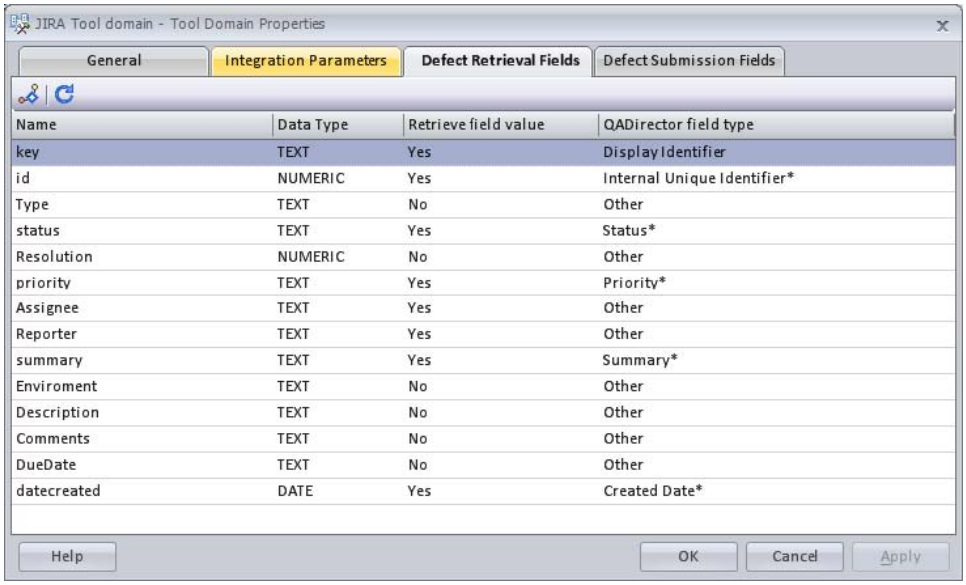
4. Create a new JIRA tool domain that belongs to the JIRA tool that was created in the previous step.



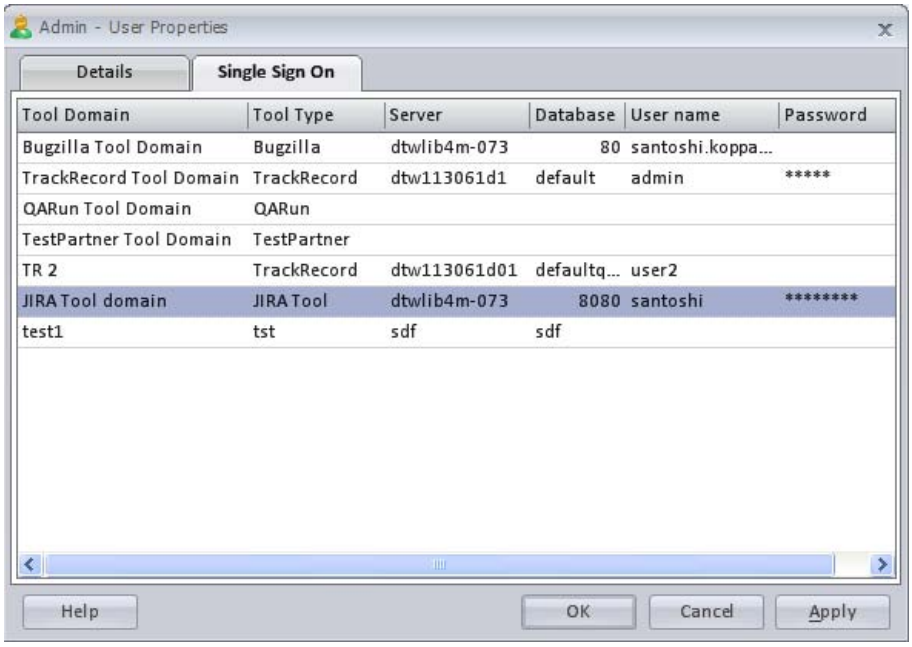
5. Select the **Integration Parameters** tab and provide the appropriate values for the integration parameters. For example, for the `Server Name` parameter, provide the JIRA server name. After providing the parameter values, save the tool domain and click the **Test tool domain** tool bar icon to test login to JIRA.



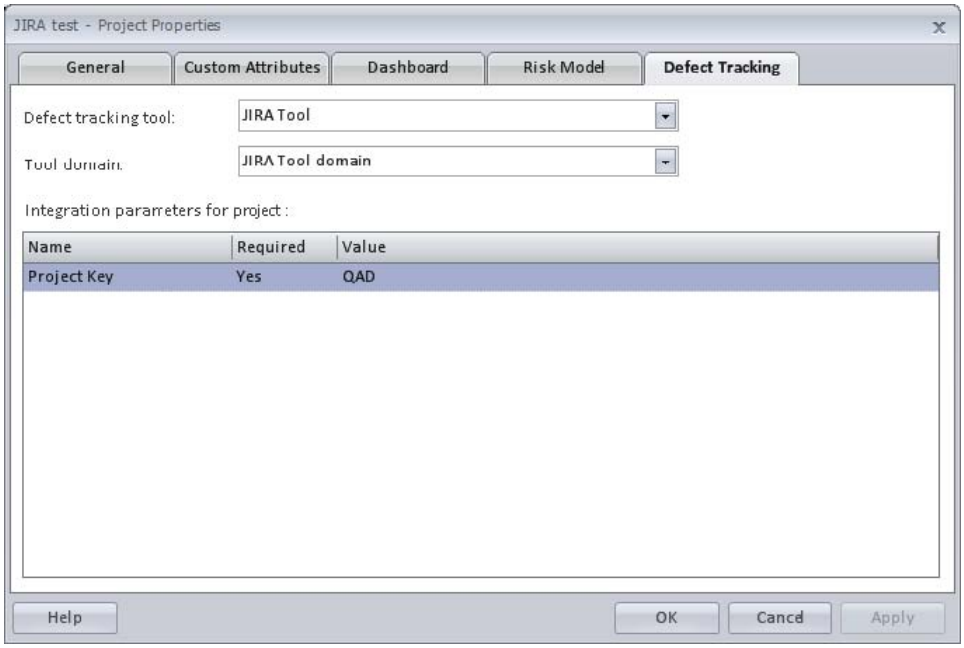
6. Select the **Defect Retrieval Fields** tab and click the **Reload** button to load JIRA's defect fields. In the **Retrieve field value** column, select **`Yes`** for the fields that you want to see in the *QA*Director **Defect Center**. Use the **Status Priority Mapping** tool bar icon to map priority and statuses (optional step). Make appropriate selections in the **Defect Submission** fields and save the tool domain.



7. Go to the **User Properties>Single Sign On** tab. Find the JIRA tool domain that was created in the previous step. For the JIRA tool domain, provide the JIRA login and password information.



8. Select the *QA*Director project of interest, go to **Project Properties>Defect Tracking** tab. Select the JIRA tool and tool domain, provide the JIRA project key to be used with the current *QA*Director project. Save the project properties.



This completes the JIRA integration set up. Defects can be submitted to *QA*Director or retrieved from JIRA to *QA*Director.

Automated Tool Integration

*QA*Director exposes an integration interface which allows you to use the automated tool of your choice. Follow the steps in this section to use any third party automated tool to create, to edit, or to retrieve scripts. See Deployment [p. 52] after your code is compiled to complete the necessary steps of your integration.

The automated tool integration points include:

- Testing the connection to the tool.
- Retrieving field names from the tool.
- Retrieving Scripts from the tool to display in *QA*Director **Global** and **Script Center.**
- Creating scripts from *QA*Director.
- Editing Scripts.
- Running Scripts.

In order to complete the integration, you need to have technical knowledge of .NET-related technologies and be able to create a .NET application.

System Requirements

- An automated testing tool.
- Visual Studio 2005 and the .NET framework 2.0 for building your integration DLL.
- Refer to the automated tool integration C# code samples installed with this SDK for a better understanding of the integration architecture.

Code Reference

Before building your custom DLL, analysis is needed to find the necessary APIs and integration points of the automated tool that will be used for *QA*Director integration. Per the *QA*Director infrastructure, the integration requires a tool and tool domain set up with an external DLL that can be invoked by QADirector.

You will need to create a .NET application with an output type of *Class Library*. *QA*Director requires that the DLL has an interface named `IThirdPartyAutomated` and a class named `ToolClass`. Another interface named `IExecutionAPI` must be used in order to connect to the *QA*Director **Test Execution Agent** during script execution.

IThirdPartyAutomated Interface

The **`IThirdPartyAutomated Interface`** declares the methods that need to be implemented in the integration application.

See ToolClass [p. 43] for an example of how to create and implement this interface.

ToolClass

This class is derived from the IThirdPartyAutomated Interface [p. 43]. All of its interface methods should be implemented in this class.

```
interface IThirdPartyAutomated
{
   string GetScriptListFromTool(string inputXML);
   string EditScript(string inputXML);
   string NewScript(string inputXML);
   string GetScriptFieldListFromTool(string inputXML);
   bool   TestConnection(string inputXML);
   bool   RunScript();
}

ToolClass: IThirdPartyAutomated
{
  string GetScriptListFromTool(string inputXML);
  { //implementation}

  string EditScript(string inputXML);
  { //implementation}

  string NewScript(string inputXML);
  { //implementation}

  string GetScriptFieldListFromTool(string inputXML);
  { //implementation}

  bool   TestConnection(string inputXML);
  { //implementation}

  bool   RunScript();
  { //implementation}

 }
```

Parameter Information

The `inputXML` parameter sends the integration parameters and their values as defined in the tool and tool domain. Depending on which method is called, the inputXML may contain additional information.

Return Value

All element names in the return XML must be exactly as they appear in the examples in this section. For example, the element **`returncode`** cannot be **`ReturnCode.`**

Member Information

EditScript

Edits a script.

Syntax

```
EditScript(string inputXML)
```

Parameters

The input string is an XML string that is auto generated by *QA*Director based on the tool and tool domain settings. The XML will contain tool and tool domain values plus the properties of the selected script.

```
<root>
   <tool>
      <field name=\"Tool Attribute1\" value=\"this is my tool value\" />
   </tool>
   <tooldomain>
      <field name=\"Database Server Name\" value=\"dbservernamehere\" />
      <field name=\"Database Name\" value=\"dbnamehere\" />
      <field name=\"Database User Name\" value=\"dbusernamehere\" />
      <field name=\"Database User Password\" value=\"Oc+hog4CqPNRjIvHQeErAg==\" />
      <field name=\"Other value\" value=\"other value here\" />
   </tooldomain>
   <script name=\"script1\" description=\"script1Desc\">
      <field name=\"field1\" value=\"field1value\" />
      <field name=\"field2\" value=\"field2value\" />
   </script>
</root>
```

Type

```
string
```

GetScriptFieldListFromTool

Retrieves the fields information of the tool.

Syntax

```
GetScriptFieldListFromTool(string inputXML)
```

Parameters

The input string is an xml string that is auto generated by QADirector based on the tool and tool domain settings. The XML will contain only tool and tooldomain values.

```
<root>
   <tool>
      <field name=\"Tool Attribute1\" value=\"this is my tool value\" />
   </tool>
   <tooldomain>
      <field name=\"Database Server Name\" value=\"dbservernamehere\" />
      <field name=\"Database Name\" value=\"dbnamehere\" />
      <field name=\"Database User Name\" value=\"dbusernamehere\" />
      <field name=\"Database User Password\" value=\"Oc+hog4CqPNRjIvHQeErAg==\" />
      <field name=\"Other value\" value=\"other value here\" />
   </tooldomain>
</root>
```

Return Value

Returns a string field list from tool that will be selected or deselected to be viewed in the **Script Center**.

```
<root>
   <fields>
      <field name="Assignment" selected="0"  />
      <field name="Category"  selected="0" />
   </fields>
</root>
```

GetScriptListFromTool

Gets the Script list from the tool.

Syntax

```
GetScriptListFromTool(string inputXML)
```

Parameters

The input string is an xml string that is auto generated by *QA*Director based on the tool and tool domain settings. The XML will contain only tool and tooldomain values.

```
<root>
   <tool>
      <field name=\"Tool Attribute1\" value=\"this is my tool value\" />
   </tool>
   <tooldomain>
      <field name=\"Database Server Name\" value=\"dbservernamehere\" />
      <field name=\"Database Name\" value=\"dbnamehere\" />
      <field name=\"Database User Name\" value=\"dbusernamehere\" />
      <field name=\"Database User Password\" value=\"Oc+hog4CqPNRjIvHQeErAg==\" />
      <field name=\"Other value\" value=\"other value here\" />
   </tooldomain>
</root>
```

Return Value

XML string

```
<root>
   <script name="s1" description = "s1desc">
      <field name="This is my tool field1" value="This is my value" />
      <field name="This is my tool field2" value="This is my other value" />
   </>

   <script name="s2" description = "s2desc">
```

```
      <field name="This is my tool field1" value="This is my value" />
      <field name="This is my tool field2" value="This is my other value" />
   </>
</root>
```

NewScript

Creates a new script.

Syntax

```
NewScript(string inputXML)
```

Parameters

The input string is an XML string that is auto generated by *QA*Director based on the tool and tool domain settings. The XML will contain only tool and tooldomain values.

```
<root>
   <tool>
      <field name=\"Tool Attribute1\" value=\"this is my tool value\" />
   </tool>
   <tooldomain>
      <field name=\"Database Server Name\" value=\"dbservernamehere\" />
      <field name=\"Database Name\" value=\"dbnamehere\" />
      <field name=\"Database User Name\" value=\"dbusernamehere\" />
      <field name=\"Database User Password\" value=\"Oc+hog4CqPNRjIvHQeErAg==\" />
      <field name=\"Other value\" value=\"other value here\" />
   </tooldomain>
</root>
```

Return Value

string

RunScript

This method must be used in order to run a script. From within this method, a connection should be made to the *QA*Director **Test Execution Agent** using the IExecutionAPI Interface [p. 50].

Syntax

```
RunScript()
```

Example

Use the following code to make a connection to the *QA*Director **Test Execution Agent** using the IExecutionAPI Interface [p. 50]:

```
IExecutionAPI eiAPI;
eiAPI = (IExecutionAPI)Activator.GetObject(typeof(IExecutionAPI),
"ipc://IntegrationCommunicationServer/IntegrationCommunicationServer.rem");
```

For more information, see Running a Script Example [p. 49].

TestConnection

Tests the connection to the tool by using the integration parameters provided in the tool domain.

Syntax

`TestConnection (string inputXML)`

Parameters

The input string is an xml string that is auto-generated by *QA*Director based on the tool and tool domain settings. The XML will contain only tool and tooldomain values.

```
<root>
   <tool>
      <field name=\"Tool Attribute1\" value=\"this is my tool value\" />
   </tool>
   <tooldomain>
      <field name=\"Database Server Name\" value=\"dbservernamehere\" />
      <field name=\"Database Name\" value=\"dbnamehere\" />
      <field name=\"Database User Name\" value=\"dbusernamehere\" />
      <field name=\"Database User Password\" value=\"Oc+hog4CqPNRjIvHQeErAg==\" />
      <field name=\"Other value\" value=\"other value here\" />
   </tooldomain>
</root>
```

Return Value

Returns a bool of True or False.

Getting Scripts Example

```
public string GetScriptListFromTool(string inputXML)
{
   //Build the xml. Input xml is expected to be in this format
   /*
   <root>
   <tool>
      <field name="This is my tool field1" value=”This is my value” />
      <field name="This is my tool field2" value=”This is my other value” />
   </tool>
   <tooldomain>
      <field name="This is my tool domain field1" value=”This is my value” />
      <field name="This is my tool domain field2" value=”This is my other value” />
   </tooldomain>
   <script>
      <field name="This is my script property field1" value=”This is my value” />
     <field name="This is my script property field2" value=”This is my other value” />

   </script>
   </root>
   /* return XML expected in this format
   <root>
      <script name=”s1” description = “s1desc”>
         <field name="This is my tool field1" value=”This is my value” />
         <field name="This is my tool field2" value=”This is my other value” /></>
      <script name=”s2” description = “s2desc”>
         <field name="This is my tool field1" value=”This is my value” />
         <field name="This is my tool field2" value=”This is my other value” /></>
   </root>
   */

   XmlDocument xmlDoc = new XmlDocument();
   XmlElement elMain = xmlDoc.CreateElement("root");
   xmlDoc.AppendChild(elMain);
   XmlElement newScriptElem;
   newScriptElem = this.CreateScriptNode(xmlDoc, "script1", "script1Desc");
   elMain.AppendChild(newScriptElem);
   XmlElement newFieldElem;
   newFieldElem = this.CreateFieldNode(xmlDoc, "f1", "feild1val");
   newScriptElem.AppendChild(newFieldElem);
   XmlElement newFieldElem2;
   newFieldElem2 = this.CreateFieldNode(xmlDoc, "f2", "feild1val2");
   newScriptElem.AppendChild(newFieldElem2);
```

```
   XmlElement newFieldElem3;
   newFieldElem3 = this.CreateFieldNode(xmlDoc, "f3", "feild1val2");
   newScriptElem.AppendChild(newFieldElem3);
   XmlElement newFieldElem4;
   newFieldElem4 = this.CreateFieldNode(xmlDoc, "f4", "feild1val2");
   newScriptElem.AppendChild(newFieldElem4);
   XmlElement newFieldElem5;
   newFieldElem5 = this.CreateFieldNode(xmlDoc, "f5", "feild1val2");
   newScriptElem.AppendChild(newFieldElem5);
   XmlElement newFieldElem6;
   newFieldElem6 = this.CreateFieldNode(xmlDoc, "f6", "feild1val2");
   newScriptElem.AppendChild(newFieldElem6);
   // script 2
   XmlElement newScriptElem1;
   newScriptElem1 = this.CreateScriptNode(xmlDoc, "script2", "script2Desc");
   elMain.AppendChild(newScriptElem1);
   XmlElement newFieldElem1;
   newFieldElem1 = this.CreateFieldNode(xmlDoc, "f1", "feild2val");
   newScriptElem1.AppendChild(newFieldElem1);
   XmlElement newFieldElem7;
   newFieldElem7 = this.CreateFieldNode(xmlDoc, "f2", "feild2val");
   newScriptElem1.AppendChild(newFieldElem7);
   return xmlDoc.InnerXml;
}
```

Getting Script Field List from Tool Example

```
public string GetScriptFieldListFromTool(string inputXML)
{

   /*<root>
   <tool>
      <field name="This is my tool field1" value=”This is my value” />
      <field name="This is my tool field2" value=”This is my other value” />
   </tool>
   <tooldomain>
      <field name="This is my tool domain field1" value=”This is my value” />
      <field name="This is my tool domain field2" value=”This is my other value” />
   </tooldomain>
      <script>
         <field name="This is my script property field1" value=”This is my value” />
        <field name="This is my script property field2" value=”This is my other value”
 />
      </script>
   </root>
   /* Return field list from tool that will be selected or deselected to be viewed in
Script Center
   <root>
      <fields>
         <field name="Assignment" selected="0" />
         <field name="Category" selected="0" />
      </fields>
   <root>
   inputXML gets the Tool Properties and Tool Domain
   */

   XmlDocument xmlDoc = new XmlDocument();
   XmlElement elMain = xmlDoc.CreateElement("root");
   xmlDoc.AppendChild(elMain);
   XmlElement newFieldsElem;
   newFieldsElem = xmlDoc.CreateElement("fields");
   elMain.AppendChild(newFieldsElem);
   XmlElement newFieldElem;
   newFieldElem = this.CreateRetrievalFieldNode(xmlDoc,"f1","0");
   newFieldsElem.AppendChild(newFieldElem);
   XmlElement newFieldElem1;
   newFieldElem1 = this.CreateRetrievalFieldNode(xmlDoc, "f2", "0");
   newFieldsElem.AppendChild(newFieldElem1);
   XmlElement newFieldElem2;
   newFieldElem2 = this.CreateRetrievalFieldNode(xmlDoc, "f3", "0");
   newFieldsElem.AppendChild(newFieldElem2);
   XmlElement newFieldElem3;
   newFieldElem3 = this.CreateRetrievalFieldNode(xmlDoc, "f4", "0");
```

```
   newFieldsElem.AppendChild(newFieldElem3);
   XmlElement newFieldElem4;
   newFieldElem4 = this.CreateRetrievalFieldNode(xmlDoc, "f5", "0");
   newFieldsElem.AppendChild(newFieldElem4);
   XmlElement newFieldElem5;
   newFieldElem5 = this.CreateRetrievalFieldNode(xmlDoc, "f6", "0");
   newFieldsElem.AppendChild(newFieldElem5);
   return xmlDoc.InnerXml;
```

Running a Script Example

The **`RunScript`** method must be used in order to run a script. From within this method, a connection should be made to the *QA*Director **Test Execution Agent** using the IExecutionAPI Interface [p. 50].

```
public bool RunScript()
{
   string detailFileData = "";

   //Make a connection to the QADirector Test Execution Agent
   IExecutionAPI eiAPI;
   eiAPI = (IExecutionAPI)Activator.GetObject(typeof(IExecutionAPI),
   "ipc://IntegrationCommunicationServer/IntegrationCommunicationServer.rem");

   //GetScriptParameters returns the Tool, Tool Domain, and current script definitions
   string inputXML = eiAPI.GetScriptParameters();

   /*
    * Script Execution logic here
    */

   //Example of passing a script
   bool bScriptStatus = true;

   if (bScriptStatus)
   {
      detailFileData = "This test passed.";
      eiAPI.SetResultOutcome(true);
   }
   else
   {
      detailFileData = "This test failed.";
      eiAPI.SetResultOutcome(false);
      eiAPI.SetResultString(detailFileData);
   }

   //Setting result file example
   System.Text.ASCIIEncoding encoding = new System.Text.ASCIIEncoding();
   byte[] fileBuffer = encoding.GetBytes(inputXML);

   eiAPI.SetResultFile("passed in data.xml", fileBuffer, false);

  //Setting detail result file example - Setting "isDetail" to true enables the Detail
 button
   //from the Script Result Properties dialog.
   fileBuffer = encoding.GetBytes(detailFileData);
   eiAPI.SetResultFile("detail.txt", fileBuffer, true);

   /* Adding a binary file example
   * This example opens a FileStream to a binary file and uses the BinaryReader
   * to read the bytes of the file.  Use this instead of the ASCIIEncoding when dealing
 with binary files.
   */
   string fileName = @"c:\My Directory\My Document.doc";

   //Use to get the length of the file
   System.IO.FileInfo fInfo = new System.IO.FileInfo(fileName);

   System.IO.FileStream objStreamQAD = new System.IO.FileStream(fileName,
FileMode.OpenOrCreate);
   System.IO.BinaryReader binaryReader = new System.IO.BinaryReader(objStreamQAD);
   fileBuffer = binaryReader.ReadBytes((int)fInfo.Length);
```

```
   eiAPI.SetResultFile("my document.doc", fileBuffer, true);

   return true;
}
```

IExecutionAPI Interface

This interface must be used in order to pass information to the *QA*Director **Test Execution Agent** during script execution. It must be used in the definition of the RunScript [p. 46] method in the ToolClass [p. 43].

```
namespace Compuware.QACenter.QADirector.ExecutionInterface
{
   public interface IExecutionAPI
   {
      string GetScriptParameters();
      void SetResultFile(string fileName, byte[] resultFile, bool isDetail);
      void SetResultOutcome(bool resultOutcome);
      void SetResultString(string resultString);
   }
}
```

- GetScriptParameters [p. 50]
- SetResultFile [p. 50]
- SetResultOutcome [p. 51]
- SetResultString [p. 51]

GetScriptParameters

Gets information on the current script and its associated tool and tool domain.

Syntax

```
GetScriptParameters()
```

Return Value

The output string is an xml string that is auto generated by *QA*Director based on the tool and tool domain settings. The XML will contain tool and tool domain values plus the properties of the selected script.

```
<root>
   <tool>
      <field name=\"Tool Attribute1\" value=\"this is my tool value\" />
   </tool>
   <tooldomain>
      <field name=\"Database Server Name\" value=\"dbservernamehere\" />
      <field name=\"Database Name\" value=\"dbnamehere\" />
      <field name=\"Database User Name\" value=\"dbusernamehere\" />
      <field name=\"Database User Password\" value=\"Oc+hog4CqPNRjIvHQeErAg==\" />
      <field name=\"Other value\" value=\"other value here\" />
   </tooldomain>
   <script name=\"script1\" description=\"script1Desc\">
      <field name=\"field1\" value=\"field1value\" />
      <field name=\"field2\" value=\"field2value\" />
   </script>
</root>
```

SetResultFile

Used to pass a file back to the *QA*Director **Test Execution Agent** which can then be viewed within *QA*Director's results.

Syntax

```
SetResultFile(string fileName, byte[] resultFile, bool isDetail)
```

Parameters

- `string fileName` - The name of the file. This file name will appear within *QA*Director results for this particular script.
- `byte[] resultFile` - A byte array that represents the file's contents.
- `bool isDetail` - If this is set to true, when the results are viewed in *QA*Director's **Job Results Detail**, if this script is double-clicked within the user interface, it will try to launch this file. Only one detail file can be set per script per execution. When set to false, this file will appear on the **Result Summary** window in *QA*Director **Job Results Detail**.

Return Value

`void`

SetResultOutcome

Used to pass or fail the current script that is running.

Syntax

```
SetResultOutcome(bool resultOutcome)
```

Parameters

`bool resultOutcome` - Set to true to pass the script. Set to false to fail the script.

Return Value

`void`

SetResultString

Used to pass a failure message in the event that a script fails. This should be used in conjunction with SetResultOutcome [p. 51], where the result outcome was a failed script. The failure message will appear in *QA*Director's **Job Result Detail**.

Syntax

```
SetResultString(string resultString)
```

Parameters

`string resultString` - The failure message for the script.

Return Value

`void`

Deployment

1. After coding the integration DLL, the output binaries should be copied to the following location on the *QA*Director web server machine: `\Micro Focus\QADirector\TMServices\ThirdPartyIntegrations`. Your DLL will be deployed in the following manner:

   Automated Tools

   Your DLL will be downloaded from the server to the client when needed.

   Defect Tools

   - For defect retrieval, the DLL is executed on the server.
   - For defect editing, the DLL is always downloaded and executed on the client on demand.
   - For defect submission, the default behavior is silent submission on the server. However, the **Tool Properties** dialog box provides an option to submit defects on the client. If this option is selected, then the DLL will be download and executed on the client.

2. Create a **Tool** and **Tool Domain** in *QA*Director to allow *QA*Director to send and receive the appropriate information.

   - When creating a **Tool**, be sure to select the appropriate type from the **Tool Type** list. For example: for defects, select the **`Defect`** type. For automated tools, select the **`Automated`** type.
   - The **Tool/Tool Domain** integration parameters are created and configured based on the needs of the tool/integration:

     Automated Tools

     Parameters can be set to apply at the tool domain or the tool properties. By applying a parameter at the tool properties, a value can be set that can be used in the tool. Additional Parameters can be added.

     Defect Tools

     Parameters for defect tool domains can be set to apply at the tool domain or the project level. By applying a parameter at the project level, duplicate tool domains can be avoided. The integration parameter values are set in two locations. Parameters that apply across the tool domain are set in the tool domain properties integration parameters tab, while parameters that apply to projects are set in the project properties defect tracking tab.

   - **Single Sign On**

     Automated Tools

     You can optionally use **Single Sign On** for the integration.

     Defect Tools

     Single Sign On is required for Defect submission and editing. They use the defect tool login specified in single sign on.

For more information on Single Sign On and *QA*Director/third-party integrations, search for the following topic in the *QA*Director online help: *Integrating with External Products*.

- For Defect Tool integrations, be sure to associate the Project with the Tool Domain.

CHAPTER 3

*QA*Director API

The *QA*Director API provides a robust set of classes to help you to create custom integrations into *QA*Director. The SDK includes a sample application that leverages many of the classes contained within the API.

Unless otherwise noted, the *QA*Director API throws exceptions if errors occur. Make sure you wrap these calls to the API with structured error handling. For example:

```
private TreeNode GetProjectsForClient(TreeNode ProjectsTreeNode, Client QADClient)
{
     try
     {
          foreach (Project QADProject in QADClient.Projects)
          {
               ProjectsTreeNode.Nodes.Add(new TreeNode(QADProject.Name));
          }
      }
      catch (Exception ex) { MessageBox.Show(ex.Message.ToString()); }

      return ProjectsTreeNode;
}
```

System Requirements

- *QA*Director API (`TMClient.dll`).
- Visual Studio 2005/.NET 2.0.
- Refer to the C# sample application installed with this SDK for a better understanding of the integration architecture.

Deployment

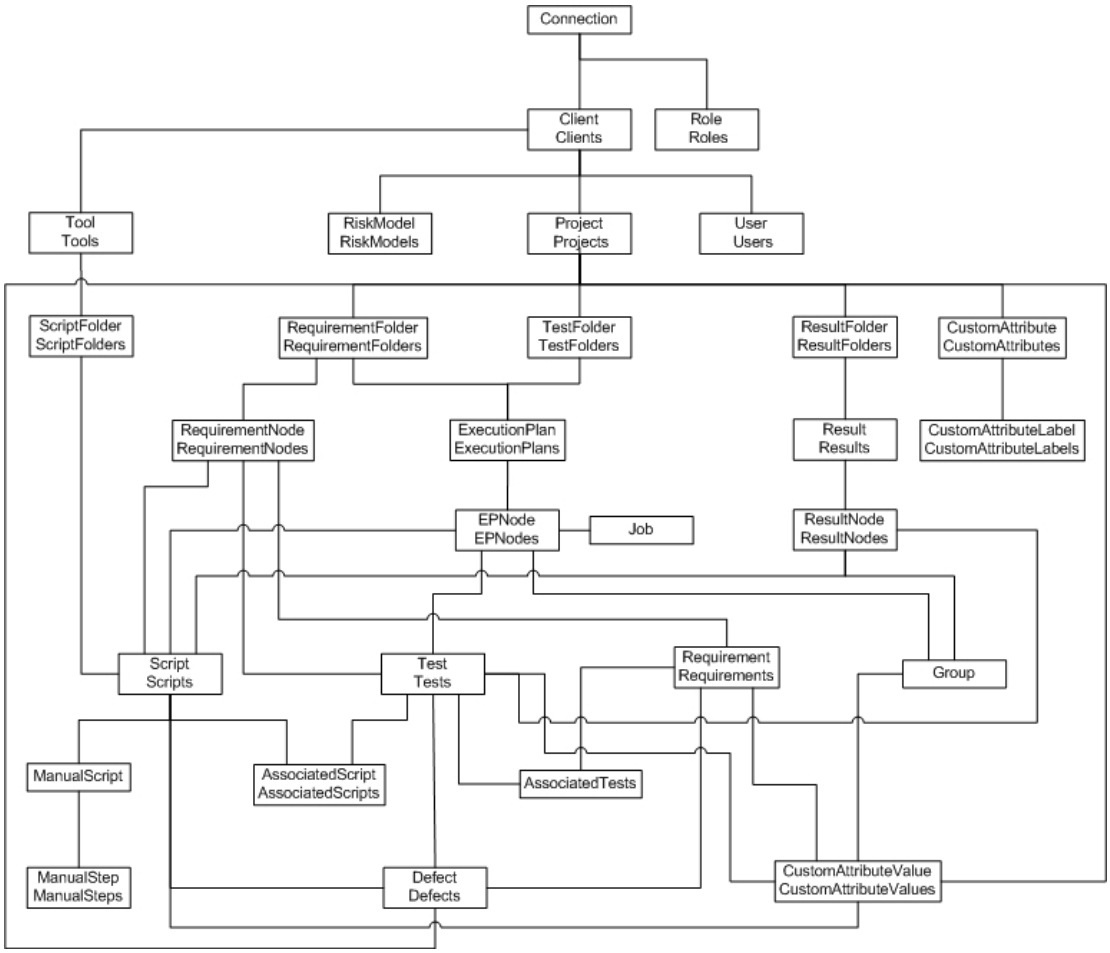
When you deploy your applications created with this SDK, be sure to deploy the following files:

1. `TMClient.dll`
2. `Localization.dll`

API Class Diagram

The following is a conceptual diagram and does not show the exact class relationships of the *QA*Director API.

Figure 1. *QA*Director API Class Diagram



API Reference

General Classes

AssociatedScript

Returns an `AssociatedScript` object. An associated script is a script that has an association to a Test.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- CreatedByUser [p. 57]

Methods

Properties

CreatedByUser

Returns the User [p. 192] that created the AssociatedScript [p. 56].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.AssociatedScript`

CustomAttributeValues

Gets the CustomAttributeValues [p. 74] collection associated with the AssociatedScript [p. 56].

Type: `Compuware.QM.QADirector.SDK.CustomAttributeValues`

Namespace: `Compuware.QM.QADirector.SDK.AssociatedScript`

Description

Returns the description field of the AssociatedScript [p. 56] object.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.AssociatedScript`

ModifiedByUser

Returns the User [p. 192] that last modified the AssociatedScript [p. 56].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.AssociatedScript`

Name

Returns the name field for the AssociatedScript [p. 56].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.AssociatedScript`

OrderNo

Returns the Order No value of the AssociatedScript [p. 56]. This value indicates the AssociatedScript's relative order with sibling AssociatedScript objects.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.AssociatedScript`

Script

Returns the Script [p. 164] object of the AssociatedScript [p. 56].

Type: `Script`

Namespace: `Compuware.QM.QADirector.SDK.AssociatedScript`

ScriptDefID

Returns the ScriptDefID of the AssociatedScript [p. 56].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.AssociatedScript`

TestScriptRelID

Returns the `TestScriptRelID` field of the AssociatedScript [p. 56].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.AssociatedScript`

Methods

UpdateCAs

Updates the CustomAttributeValues [p. 74] collection contained within the AssociatedScript [p. 56].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.AssociatedScript`

Syntax

```
UpdateCAs()
```

AssociatedScripts

Returns a collection of AssociatedScript [p. 56] objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

Count [p. 59]

Methods

- Exists [p. 59]

- Refresh [p. 196]

Properties

Count

Gets the number of AssociatedScript [p. 56] objects in the AssociatedScripts [p. 58] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.AssociatedScripts`

Methods

Exists

Returns true/false whether an AssociatedScript [p. 56] exists in the AssociatedScripts [p. 58]collection.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.AssociatedScripts`

Syntax

```
Exists(int TestScriptRelID)
Exists(string scriptName)
```

Refresh

This method refreshes the item(s) from the database.

Type: `void`

Syntax

```
Refresh()
```

AssociatedTests

Returns a collection of `AssociatedTest` objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- AssociatedTests [p. 59]
- Count [p. 60]

Properties

AssociatedTests

Indexer for `AssociatedTests` that returns a Test [p. 174] object.

Type: `Test`

Namespace: `Compuware.QM.QADirector.SDK.AssociatedTests`

Syntax

```
AssociatedTests[int ID]
```

Count

Gets the number of Test [p. 174] objects in the AssociatedTests [p. 59] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.AssociatedTests`

Client

Provides access to all of the information about a given `Compuware.QM.QADirector.SDK.Client` in *QA*Director.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- CreatedByUser [p. 61]
- CustomAttributes [p. 61]
- Description [p. 61]
- ID [p. 61]
- ModifiedByUser [p. 61]
- Name [p. 61]
- Projects [p. 61]
- RiskModels [p. 62]
- TestingTools [p. 62]
- Users [p. 62]

Methods

- CloseClient [p. 62]
- OpenClient [p. 62]

Properties

ClientCAExists

Checks to see if a Custom Attribute exists in the current Client [p. 60].

Type: `bool`

Member of `Compuware.QM.QADirector.SDK.Client`

Syntax

```
ClientCAExists(int CAID)
```

int CAID
ID of the Custom Attribute

CreatedByUser

Returns the User [p. 192] that created the Client [p. 60].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Client`

CustomAttributes

Returns the CustomAttributes [p. 71] collection associated with the Client [p. 60].

Type: `Compuware.QM.QADirector.SDK.CustomAttributes`

Namespace: `Compuware.QM.QADirector.SDK.Client`

Description

Gets the Client [p. 60] description field.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Client`

ID

Gets the Client [p. 60] ID.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Client`

ModifiedByUser

Returns the User [p. 192] that last modified the Client [p. 60].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Client`

Name

Gets the Client [p. 60] name.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Client`

Projects

Returns the Projects [p. 131] collection associated with the Client [p. 60].

Type: `Compuware.QM.QADirector.SDK.Projects`

Namespace: `Compuware.QM.QADirector.SDK.Client`

RiskModels

Gets the RiskModels [p. 161] collection associated with the Client [p. 60].

Type: `Compuware.QM.QADirector.SDK.RiskModels`

Namespace: `Compuware.QM.QADirector.SDK.Client`

TestingTools

Returns the Tools [p. 192] collection associated with the Client [p. 60].

Type: `Compuware.QM.QADirector.SDK.Tools`

Namespace: `Compuware.QM.QADirector.SDK.Client`

Users

Returns the Users [p. 195] collection associated with the Client [p. 60].

Type: `Compuware.QM.QADirector.SDK.Users`

Namespace: `Compuware.QM.QADirector.SDK.Client`

Methods

CloseClient

Closes a Client [p. 60].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Client`

Syntax

```
CloseClient()
```

OpenClient

Opens a Client [p. 60].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Client`

Syntax

```
OpenClient()
```

Clients

Provides access to all of the `Clients` in *QA*Director.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 63]
- Clients [p. 63]

Methods

Refresh [p. 196]

Properties

Clients

- **`Clients[int clientid]`** gets a **`Client`** object by the supplied clientid.
- **`Clients[string clientname]`** gets a **`Client`** object by the supplied name.

Type: `Compuware.QM.QADirector.SDK.Client`

Count

Gets the count of clients.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Clients`

Methods

Refresh

This method refreshes the item(s) from the database.

Type: `void`

Syntax

`Refresh()`

Connection

Connects to **Test Management web services**. Users needs to retain a reference to access other exposed functionality.

Namespace: `Compuware.QM.QADirector.SDK`

Methods

- IsConnected [p. 65]
- LogOff [p. 65]
- LogOn [p. 65]

Properties

- APIVersion [p. 64]
- Clients [p. 64]
- CurrentClient [p. 64]
- CurrentProject [p. 64]
- CurrentUser [p. 64]

- Roles [p. 64]
- Users [p. 64]

Code Sample

Connection/Logon Example [p. 65]

Properties

APIVersion

Gets the *QA*Director API version.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Connection`

Clients

Gets the Clients [p. 62] collection of the active Connection [p. 63].

Type: `Compuware.QM.QADirector.SDK.Clients`

Namespace: `Compuware.QM.QADirector.SDK.Connection`

CurrentClient

Gets the Connection [p. 63]'s active Client [p. 60].

Type: `Compuware.QM.QADirector.SDK.Client`

Namespace: `Compuware.QM.QADirector.SDK.Connection`

CurrentProject

Gets the Connection [p. 63]'s active Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.Project`

Namespace: `Compuware.QM.QADirector.SDK.Connection`

CurrentUser

Gets the User [p. 192] object of the user logged in to the Connection [p. 63].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Connection`

Roles

Gets the Roles [p. 163] collection associated with the Connection [p. 63].

Type: `Compuware.QM.QADirector.SDK.Roles`

Namespace: `Compuware.QM.QADirector.SDK.Connection`

Users

Returns the Users [p. 195] collection associated with the Connection [p. 63].

Type: `Compuware.QM.QADirector.SDK.Users`

Namespace: `Compuware.QM.QADirector.SDK.Connection`

Connection/Logon Example

```
private void ConnectToDatabase(string user, string password, string servername, string
 virtualDirectory, bool isSSL)
{
      Connection connection = new Connection();
     bool IsLoggedIn = connection.Logon(user,password,servername,virtualDirectory,isSSL);

      if (connection.IsConnected())
      {
           MessageBox.Show("Logged in successfully.");
      }
      else
      {
           MessageBox.Show("Error connecting to QADirector.");
      }
}
```

Methods

IsConnected

Indicates if the Connection [p. 63] is still active.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Connection`

Syntax

`IsConnected()`

LogOff

Logs off a user from the Connection [p. 63].

Type: `void`

Namespace: `Compuware.QM.QADirector.SDK.Connection`

Syntax

`LogOff()`

LogOn

Authenticates a user to log on to the SDK via the Connection [p. 63].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Connection`

Syntax

`LogOn(string UserName,string Password, string serverName,string virtualDirectory, bool isSSL, Int PortNumber)`

`LogOn(string UserName,string Password, string serverName,string virtualDirectory, bool isSSL)`

string UserName
User name to connect to *QA*Director.

string Password
Password to connect to *QA*Director.

string serverName
Name of the web server to connect to.

string virtualDirectory
Name of the Virtual directory.

bool isSSL
Indicates if SSL is enabled on the server.

Int PortNumber
If your web server's default `Port Number` is not `80`, indicate the port number using the overloaded method.

CustomAttribute

Returns a `CustomAttribute` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- SecondaryOptions [p. 69]

Properties

AppliedOnAsset

Asset to which the CustomAttribute [p. 66] is applied.

Type: `Compuware.QACenter.TM.eAssetTypes`. See eAssetTypes [p. 88].

Namespace: `Compuware.QM.QADirector.SDK.AssociatedScript`

CAType

Returns the type of CustomAttribute [p. 66].

Type: `Compuware.QACenter.TM.eCAKind`. See eCAKind [p. 83].

Namespace: `Compuware.QM.QADirector.SDK.AssociatedScript`

CreatedByUser

Returns the User [p. 192] that created the CustomAttribute [p. 66].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

CustomAttributeLabels

Returns the CustomAttributeLabels [p. 70] collection associated with the CustomAttribute [p. 66].

Type: `Compuware.QM.QADirector.SDK.CustomAttributeLabels`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

Data

Returns the `DataRow` object associated with the CustomAttribute [p. 66].

Type: `System.Data.DataRow`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

DataType

Returns the data type for the CustomAttribute [p. 66].

Type: `Compuware.QACenter.TM.eCADataType`. See eCADataType [p. 83].

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

DecimalPlaces

Number of decimal places allowed for the CustomAttribute [p. 66] with `Numeric` as the data type.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

DefaultValue

Value that will be used if no value is provided for the CustomAttribute [p. 66].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

Description

Returns the description field for the CustomAttribute [p. 66].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

DisplayType

Gets the CustomAttribute [p. 66] DisplayType field.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

ID

Returns the reference ID for the CustomAttribute [p. 66].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

IsRelational

Returns the relational bool for the `CustomAttribute`.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

MaxLength

Returns the maximum length allowable for the CustomAttribute [p. 66].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

MaxVal

Maximum allowable value for a CustomAttribute [p. 66] of `Numeric` data type.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

MinVal

Minimum value allowed for a CustomAttribute [p. 66] with Numeric data type.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

ModifiedByUser

Returns the User [p. 192] that last modified the CustomAttribute [p. 66].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

Name

Returns the name of the CustomAttribute [p. 66].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

ReferenceID

Returns the `ReferenceID` field for the CustomAttribute [p. 66].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

SecondaryOptions

Returns the SecondaryOptions field for the CustomAttribute [p. 66].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttribute`

CustomAttributeLabel

Returns a `CustomAttributeLabel` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- CAID [p. 69]
- CreatedByUser [p. 69]
- Label [p. 70]
- ModifiedByUser [p. 70]
- Val [p. 70]

Properties

CAID

Get/set the Custom Attribute ID for the `CustomAttributeLabel`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeLabel`

CreatedByUser

Returns the User [p. 192] that created the CustomAttributeLabel [p. 69].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeLabel`

Label

Gets or sets the Custom Attribute label string for the CustomAttributeLabel [p. 69].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeLabel`

ModifiedByUser

Returns the User [p. 192] that last modified the CustomAttributeLabel [p. 69].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeLabel`

Val

Get/set the custom attribute `Val` for the CustomAttributeLabel [p. 69].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeLabel`

CustomAttributeLabels

Returns a collection of `CustomAttributeLabel` objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 70]
- CustomAttributeLabels [p. 70]

Properties

Count

Returns the number of CustomAttributeLabel [p. 69] objects in the CustomAttributeLabels [p. 70] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeLabels`

CustomAttributeLabels

`CustomAttributeLabels` indexer that returns a CustomAttributeLabel [p. 69] object.

Type: `CustomAttributeLabel`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeLabels`

Syntax

- **`CustomAttributeLabels[int val]`**
- **`CustomAttributeLabels[String label]`**

CustomAttributes

Returns a collection of `CustomAttribute` objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 71]
- CustomAttributes [p. 71]

Methods

- AddCustomAttributeToProject [p. 71]
- Load [p. 72]

Properties

Count

Returns the number of CustomAttribute [p. 66] objects in the CustomAttributes [p. 71] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributes`

CustomAttributes

Indexer for `CustomAttributes`.

Type: `CustomAttribute`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributes`

Syntax

`CustomAttributes[int id]`

Methods

AddCustomAttributeToProject

Adds a `CustomAttribute` from the client to the active project.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributes`

Syntax

```
AddCustomAttributeToProject(CustomAttribute ca)
```

Load

Loads CustomAttribute [p. 66] definitions and CustomAttribute [p. 66] definitions for risk models.

Type: `void`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributes`

Syntax

```
Load()
```

CustomAttributeValue

Returns a `CustomAttributeValue` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- AssetType [p. 72]
- CAID [p. 72]
- CreatedByUser [p. 73]
- ExternalID [p. 73]
- IsError [p. 73]
- LastUpdated [p. 73]
- ModifiedByUser [p. 73]
- ReqRelID [p. 73]
- ScriptID [p. 73]
- Value [p. 73]

Properties

AssetType

Returns the type of asset for the CustomAttributeValue [p. 72].

Type: `Compuware.QACenter.TM.eAssetTypes`. See eAssetTypes [p. 88].

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeValue`

CAID

Gets the reference ID for the CustomAttributeValue [p. 72].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeValue`

CreatedByUser

Returns the User [p. 192] that created the CustomAttributeValue [p. 72].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeValue`

ExternalID

Gets the `ExternalID` from the external source performing a data push for the CustomAttributeValue [p. 72].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeValue`

IsError

Returns a bool for the CustomAttributeValue [p. 72] for an error.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeValue`

LastUpdated

Gets the last date that the CustomAttributeValue [p. 72] was updated.

Type: `System.Datetime`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeValue`

ModifiedByUser

Returns the User [p. 192] that last modified the CustomAttributeValue [p. 72].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeValue`

ReqRelID

Gets a string of the related requirement ID for the CustomAttributeValue [p. 72].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeValue`

ScriptID

Gets the Script ID for a CustomAttributeValue [p. 72] that belongs to a Script.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeValue`

Value

Gets the value for the CustomAttributeValue [p. 72].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeValue`

CustomAttributeValues

Returns a collection of `CustomAttributeValue` objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

CustomAttributeValues [p. 74]

Methods

- Load [p. 74]
- New [p. 74]
- Search [p. 75]

Properties

CustomAttributeValues

Indexer for **`CustomAttributeValues`**.

Syntax

```
CustomAttributeValue this[int ID]
```

`int ID` is the id value of the **`CustomAttributeValue`** to get.

Methods

Load

Retrieves the custom attribute values for a node.

Type: `void`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributes`

Syntax

```
Load()
```

New

Adds a new CustomAttributeValue [p. 72] to the CustomAttributeValues [p. 74] collection.

Type: `CustomAttributeValue`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeValues`

Syntax

```
New(string caname, string val)
```

```
New(int caid, string val)
```

Search

Use **`Search`** to find a CustomAttributeValue [p. 72] in the CustomAttributeValues [p. 74] collection.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.CustomAttributeValues`

Syntax

- **`Search(string caname, int intScriptID)`** - Searches the collection to find a custom attribute by name that belongs to a script by id.
- **`Search(int CAID, int intScriptID)`** - Searches the collection to find a custom attribute by id that belongs to a script by id.
- **`Search(int CAID)`** - Searches the collection to find a custom attribute by id.
- **`Search(int CAID,string strExternalID)`** - Searches the collection to find a custom attribute by id and by externalid.
- **`SearchSearch(string caname, string strExternalID)`** - Searches the collection to find a custom attribute by name and by externalid.

Cycle

Returns a `Cycle` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- CreatedByUser [p. 75]
- ID [p. 75]
- ModifiedByUser [p. 76]
- Name [p. 76]
- RID [p. 76]

Properties

CreatedByUser

Returns the User [p. 192] that created the Cycle [p. 75].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Cycle`

ID

Returns the ID of the Cycle [p. 75].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Cycle`

ModifiedByUser

Returns the User [p. 192] that last modified the Cycle [p. 75].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Cycle`

Name

Returns the name of the Cycle [p. 75].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Cycle`

RID

Returns the requirment relationship id of the Cycle [p. 75].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Cycle`

CycleLabel

Returns a `CycleLabel` object. A `CycleLabel` is the name of a Cycle as defined in the **Cycles** tab of **Project Properties** dialog box.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- CreatedByUser [p. 76]
- ID [p. 76]
- ModifiedByUser [p. 77]
- Name [p. 77]
- SiblingOrder [p. 77]

Properties

CreatedByUser

Returns the User [p. 192] that created the CycleLabel [p. 76].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.CycleLabel`

ID

Returns the id of the CycleLabel [p. 76].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.CycleLabel`

ModifiedByUser

Returns the User [p. 192] that last modified the CycleLabel [p. 76].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.CycleLabel`

Name

Returns the name of the CycleLabel [p. 76].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.CycleLabel`

SiblingOrder

Returns the SiblingOrder of the CycleLabel [p. 76]. This value indicates order that the Cycles should display relative to each other.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.CycleLabel`

Cycles

Returns a collection of Cycle [p. 75] objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 77]
- Cycles [p. 77]

Methods

- GetCycleID [p. 78]
- New [p. 78]
- Refresh [p. 196]

Properties

Count

Returns the number of Cycle [p. 75] objects in the Cycles [p. 77] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Cycles`

Cycles

`Cycles` indexer that returns a Cycle [p. 75] object.

Type: `Compuware.QM.QADirector.SDK.Cycles`

Namespace: `Compuware.QM.QADirector.SDK.Cycle`

Syntax

```
Cycles[string cyclename]
Cycles[int ID]
```

Methods

GetCycleID

Returns the ID of a `Cycle` from the specified cycle name.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Cycles`

Syntax

```
GetCycleID(String cycleName)
```

New

Adds a new Cycle [p. 75] object to the `Cycles` collection with the specified name.

Type: `Compuware.QM.QADirector.SDK.Cycle`

Namespace: `Compuware.QM.QADirector.SDK.Cycles`

Syntax

```
New(string CycleName)
```

Refresh

This method refreshes the item(s) from the database.

Type: `void`

Syntax

```
Refresh()
```

CyclesLabels

Returns a collection of CycleLabel [p. 76] objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 79]
- CyclesLabels [p. 79]

Methods

- Refresh [p. 196]

Properties

Count

Returns the number of CycleLabel [p. 76] objects in the CyclesLabels [p. 78] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.CyclesLabels`

CyclesLabels

`CyclesLabels` indexer that returns a CycleLabel [p. 76] object.

Type: `Compuware.QM.QADirector.SDK.CyclesLabel`

Namespace: `Compuware.QM.QADirector.SDK.CyclesLabels`

Syntax

```
CyclesLabels[int ID]
CyclesLabels[string cycleName]
```

Methods

Refresh

This method refreshes the item(s) from the database.

Type: `void`

Syntax

```
Refresh()
```

Defect

Returns a `Defect` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- CreatedByUser [p. 80]
- DefectDefnID [p. 80]
- DefectDisplayID [p. 80]
- DefectInternalUniqueID [p. 80]
- IsAssociated [p. 80]
- ModifiedByUser [p. 80]
- Priority [p. 80]
- Status [p. 81]
- Summary [p. 81]

Properties

CreatedByUser

Returns the User [p. 192] that created the Defect [p. 79].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Defect`

DefectDefnID

Returns the DefectDefnID for the Defect [p. 79]. This field is only for associated defects. It is the unique ID for each defect association.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Defect`

DefectDisplayID

Returns the DefectDisplayID field for the Defect [p. 79]. This is the defect id that is displayed in the defect tool of *QA*Director.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Defect`

DefectInternalUniqueID

Returns the DefectInternalUniqueID for the Defect [p. 79]. This is the unique id coming from a defect tool.

`DefectInternalUniqueID` is the id used to maintain the `Defects` collection within the API. Whenever accessing a particular `Defect` from the collection, use `DefectInternalUniqueID`. For example:

```
Test.AssociatedDefects[<DefectInternalUniqueID>]
```

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Defect`

IsAssociated

Returns a bool indicating whether or not the Defect [p. 79] is associated.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Defect`

ModifiedByUser

Returns the User [p. 192] that last modified the Defect [p. 79].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.ModifiedByUser`

Priority

Returns the `Priority` of the Defect [p. 79].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Defect`

Status

Returns the status of the Defect [p. 79].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Defect`

Summary

Returns the summary information for the Defect [p. 79].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Defect`

Defects

Returns a collection of `Defect` objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 81]
- Defects [p. 81]

Methods

- Exists [p. 82]
- GetDefect [p. 82]
- Refresh [p. 196]

Properties

Count

Returns the number of Defect [p. 79] objects in the Defects [p. 81]collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Defects`

Syntax

```
Count
```

Defects

`Defects` indexer that returns a Defect [p. 79] object.

Type: `Defect`

Namespace: `Compuware.QM.QADirector.SDK.Defects`

Syntax

```
Defects[int ID]
```

Methods

Exists

Checks for the existence of a Defect [p. 79] in the Defects [p. 81] collection by ID.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Defects`

Syntax

```
Exists(string DefectToolInternalUniqueID)
```

GetDefect

Returns a Defect [p. 79] object from the Defects [p. 81] collection with the specified id.

Type: `Defect`

Namespace: `Compuware.QM.QADirector.SDK.Defects`

Syntax

```
GetDefect(string DefectToolDisplayID)
```

Refresh

This method refreshes the item(s) from the database.

Type: `void`

Syntax

```
Refresh()
```

Enums Supporting the API

The following enumerations are used to support different classes within the *QA*Director API:

Custom Attributes

- eCADataType [p. 83]
- eCAKind [p. 83]

Groups

- eExecGroupRunInParallel [p. 84]
- ExecGroupMode [p. 84]

Jobs/Results

ManualStep

Shared

Custom Attributes

eCADataType

This enum contains the values for data types.

Type: `enum`

Namespace: `Compuware.QACenter.TM`

Values

- `Numeric`
- `DateTime`
- `List`
- `Text`

eCAKind

This enum contains the values for the types of custom attributes.

Type: `enum`

Namespace: `Compuware.QACenter.TM`

Values

- `Standard`
- `Formula`
- `UserDefined`

Groups

eExecGroupRunInParallel

This `enum` contains the values for how an execution `Group` runs.

Type: `enum`

Namespace: `Compuware.QACenter.TM`

Values

- `ChildTestsOnly`
- `ChildGroupsOnly`
- `ChildGroupsAndTests`
- `None`

ExecGroupMode

This enum contains the values for execution `Group` modes.

Type: `enum`

Namespace: `Compuware.QACenter.TM`

Values

- `IsOnline`
- `IsOffline`

Jobs and Results

eExecType

This enum contains the values for Job [p. 102] types.

Type: `enum`

Namespace: `Compuware.QACenter.TM`

Values

- `Automated`
- `Manual`

eJobStatus

This enum contains the values for the different `Job` statuses in *QA*Director.

Type: `enum`

Namespace: `Compuware.QACenter.TM`

Values

- `Initializing`
- `Initialized`
- `ToBeRun`
- `TryingToRun`
- `StartingDriver`
- `Running`
- `Finished`
- `NotRunnable`
- `Suspended`
- `AbnormalExit`
- `TryingToAbort`
- `Waiting`

ScheduleOptions

eDailyOptions

This enum should be used with the `Interval` value to run recurring jobs every X days.

Type: `enum`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

Values

- `EveryXDay`
- `EveryWeekday`

eMonthlyOptions

Used with `eScheduleTypes.Monthly` recurrence.

Type: `enum`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

Values

- `SpecificDay` - This value should be used with the `Interval` value to set every X months and `DayOfMonth` value to set which day of the month recurring jobs should run.

- `OrdinalDayofWeek` - Use with `Interval` value to set every X months, OrdinalOption value to set the 1st, 2nd, 3rd, or 4th week, and set DayOfTheWeek to set which day of the week the recurring jobs should run.

eOrdinals

Used to set ordinal values.

Type: `enum`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

Values

- `First`
- `Second`
- `Third`
- `Fourth`

eSchedRangeRecur

Used in the **Job Description** to specify the range for a reocurring job.

Type: `enum`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

Values

- `EndDate` - Reocurring job that stops reocurring at a set end date.
- `NoEndDate` - Endless reocurring job.
- `NumberOfOccur` - Reocurring job that runs X number of times.

eScheduleTypes

Use the`eScheduleTypes` enum to specifiy the type of recurrence at which a job will run.

Type: `enum`

Namespace: `Compuware.QACenter.TM`

Values

- `TimeFrequency`
- `Daily`
- `Weekly`
- `Monthly`
- `Yearly`

eTimeOptions

Used with `eScheduleTypes.TimeFrequency` recurrence.

Type: `enum`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

Values

- `EveryXMinute` - Use with the `Interval` value to run recurring jobs every X minutes.
- `EveryXHour` - Use with the `Interval` value to run recurring jobs every X hours.

eYearlyOptions

Used with `eScheduleTypes.Yearly` recurrence.

Type: `enum`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

Values

- `MonthOption` - Use this option with the `Interval` value to set a specific day of the year to run recurring jobs. Also set the `Month` value in `AdditionalOptions` to indicate which month to run the job.
- `DayofWeekOption` - Use this option with `OrdinalOption` and `DayOfTheWeek` option to run a recurring job on the1st, 2nd, 3rd, or 4th day of the week each year.

Months

This enum should be used to select a month for monthly options.

Type: `enum`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

Values

- `January`
- `February`
- `March`
- `April`
- `May`
- `June`
- `July`
- `August`
- `September`
- `October`
- `November`
- `December`

Manual Steps

PassFail.CorrectAnswer

This enum contains the values for the different `PassFail.CorrectAnswer` values for ManualSteps.

Type: `enum`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep.PassFail`

Values

- `Pass = 0`
- `Fail = 1`

TrueFalse.CorrectAnswer

This enum contains the values for the different `TrueFalse.CorrectAnswer` values for ManualSteps.

Type: `enum`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep.TrueFalse`

Values

- `True = 0`
- `False = 1`

YesNo.CorrectAnswer

This enum contains the values for the different `YesNo.CorrectAnswer` values for ManualSteps.

Type: `enum`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep.YesNo`

Values

- `Yes = 0`
- `No = 1`

Shared Enums

eAssetStatus

This enum contains the values for the different asset statuses in *QA*Director.

Type: `enum`

Namespace: `Compuware.QACenter.TM`

Values

- `Complete`
- `In Progress`

eAssetTypes

This enum contains the values for the different types of assets in *QA*Director.

Type: `enum`

Namespace: `Compuware.QACenter.TM`

Values

- `Project`
- `TestRequirement`
- `Test`
- `Script`
- `ExecutionPlan`
- `ExecutionGroup`
- `RequirementFolder`
- `User`
- `Client`
- `TestCenterFolder`
- `Defect`
- `ResultFolder`
- `DefectFolder`
- `ScriptFolder`
- `ToolDomains`
- `TestFolder`
- `BuiltInToolsSearchFolders`
- `CustomToolsSearchFolders`
- `Report`
- `GlobalSearchFolders`
- `Tool`

EPNode

Returns an `EPNode` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Asset [p. 90]
- AssetType [p. 90]
- ChildNodes [p. 90]
- CreatedByUser [p. 90]
- EORelationID [p. 90]

- ExecPlan [p. 90]
- ModifiedByUser [p. 90]
- Name [p. 91]
- ParentNode [p. 91]

Properties

Asset

Returns the TMAsset [p. 187] object associated with the EPNode [p. 89].

Type: `Compuware.QM.QADirector.SDK.TMAsset`

Namespace: `Compuware.QM.QADirector.SDK.EPNode`

AssetType

Returns the type of asset for the EPNode [p. 89].

Type: `Compuware.QACenter.TM.eAssetTypes`. See eAssetTypes [p. 88].

Namespace: `Compuware.QM.QADirector.SDK.EPNode`

ChildNodes

Returns the `EPNodes` collection of children nodes of the current EPNode [p. 89].

Type: `Compuware.QM.QADirector.SDK.EPNodes`

Namespace: `Compuware.QM.QADirector.SDK.EPNode`

CreatedByUser

Returns the User [p. 192] that created the EPNode [p. 89].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.EPNode`

EORelationID

Returns the EORelationID for the EPNode [p. 89]. This is the execution order relationship id that indicates a node's relative position to its siblings.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.EPNode`

ExecPlan

Returns the ExecutionPlan [p. 91] associated with the EPNode [p. 89].

Type: `Compuware.QM.QADirector.SDK.ExecutionPlan`

Namespace: `Compuware.QM.QADirector.SDK.EPNode`

ModifiedByUser

Returns the User [p. 192] that last modified the EPNode [p. 89].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.EPNode`

Name

Returns the name of the EPNode [p. 89].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.EPNode`

ParentNode

Returns the parent `EPNode` of the current EPNode [p. 89].

Type: `Compuware.QM.QADirector.SDK.EPNode`

Namespace: `Compuware.QM.QADirector.SDK.EPNode`

EPNodes

Returns a collection of EPNode [p. 89] objects. The `EPNodes` class is used with `EPNode` and `ExecutionPlan` to return all execution plans from a project.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 91]
- EPNodes [p. 91]

Properties

Count

Returns the number of EPNode [p. 89] objects in the EPNodes [p. 91] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.EPNodes`

EPNodes

`EPNodes` indexer that returns an EPNode [p. 89] object.

Type: `EPNode`

Namespace: `Compuware.QM.QADirector.SDK.EPNodes`

Syntax

```
EPNodes[int ID]
EPNodes[string name]
```

ExecutionPlan

Returns an `ExecutionPlan` object. Inherits from TMAsset [p. 187].

Namespace: `Compuware.QM.QADirector.SDK`

Properties

Methods

Properties

AssetDefnID

Gets the asset ID value of the TMAsset [p. 187].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

AssetType

Returns the type of asset for the TMAsset [p. 187].

Type: `Compuware.QACenter.TM.eAssetTypes`. See eAssetTypes [p. 88].

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

AutomatedScriptCount

Returns the number of automated scripts in the ExecutionPlan [p. 91].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

CreatedByUser

Returns the `User` that created the ExecutionPlan [p. 91].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

CustomAttributes

Returns the CustomAttributes [p. 71] collection associated with the ExecutionPlan [p. 91].

Type: `Compuware.QM.QADirector.SDK.CustomAttributes`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

CustomAttributeValues

Returns the CustomAttributeValues [p. 74] collection associated with the ExecutionPlan [p. 91].

Type: `Compuware.QM.QADirector.SDK.CustomAttributeValues`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

DateCreated

Returns the date that the ExecutionPlan [p. 91] was created.

Type: `DateTime`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

DateModified

Returns the date that the ExecutionPlan [p. 91] was last modified.

Type: `DateTime`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

Description

Returns the description field for the ExecutionPlan [p. 91].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

DisplayID

Gets the `DisplayID` field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

EPID

Returns the ID of the `ExecutionPlan`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

EstimatedTime

Returns the estimated time in hours of the ExecutionPlan [p. 91].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

ExecutionGroupsCount

Returns the number of execution `Groups` associated with the ExecutionPlan [p. 91].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

ManualScriptCount

Returns the number of `ManualScript` objects associated with the ExecutionPlan [p. 91].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

ModifiedByUser

Returns the `User` that last modified the ExecutionPlan [p. 91].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

Name

Gets the `Name` field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

ReqFolder

Returns the RequirementFolder [p. 138] of the ExecutionPlan [p. 91].

Type: `Compuware.QM.QADirector.SDK.RequirementFolder`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

Requirements

Returns the Requirements [p. 149] associated with the ExecutionPlan [p. 91].

Type: `Compuware.QM.QADirector.SDK.Requirements`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

RequirementsCount

Returns the number of `Requirements` associated with the ExecutionPlan [p. 91].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

ResultFolder

Returns the ResultFolder [p. 154] associated with the ExecutionPlan [p. 91].

Type: `Compuware.QM.QADirector.SDK.ResultFolder`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

RootEPNode

Returns the root EPNode [p. 89] of the ExecutionPlan [p. 91].

Type: `Compuware.QM.QADirector.SDK.EPNode`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

ScriptCount

Returns the total number of Scripts associated with this ExecutionPlan [p. 91].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

Methods

AddNodeToExecutionList

Use `AddNodeToExecutionList` to add a node to the execution list. When creating a job for execution, this list can be used so that only certain nodes in the ExecutionPlan [p. 91] are run instead of the entire `ExecutionPlan`.

Type: `void`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

Syntax

`AddNodeToExecutionList(EPNode epNode)` - `epNode` must be an EPNode [p. 89] that is part of this current `ExecutionPlan`. EPNodes that are not part of this `ExecutionPlan` will be ignored.

CreateJob

Creates a new instance of a Job [p. 102] which can be used to execute an ExecutionPlan [p. 91].

Type: `Compuware.QM.QADirector.SDK.Job`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

Syntax

```
CreateJob(bool bRunEntireExecutionPlan)
```

if `bRunEntireExecutionPlan` is `true`, the returned `Job` will be set to run the entire `ExecutionPlan`. If `false`, it will only run the nodes created from the `AddNodeToExecutionList` method.

RemoveNodeFromExecutionList

Use`RemoveNodeFromExecutionList` to remove a node from the execution list. When creating a job for execution, this list can be used so that only certain nodes in the `ExecutionPlan` are run instead of the entire `ExecutionPlan`.

Type: `void`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

Syntax

```
RemoveNodeFromExecutionList(EPNode epNode)
```

UpdateCAs

Updates the custom attributes associated with the `ExecutionPlan`.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlan`

ExecutionPlans

Returns a collection of `ExecutionPlan` objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 97]
- ExecutionPlans [p. 97]

Methods

- Delete [p. 97]
- Exists [p. 97]
- Load [p. 98]
- Refresh [p. 196]
- RMNewEP [p. 98]

- RMReplaceEP [p. 98]
- RMUpdateEP [p. 99]

Properties

Count

Returns the number of ExecutionPlan [p. 91] objects associated with the Project in the ExecutionPlans [p. 96] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlans`

Syntax

```
Count
```

ExecutionPlans

Indexer for `ExecutionPlans` collection to return the specified `ExecutionPlan` object.

Type: `ExecutionPlan`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlans`

Syntax

```
ExecutionPlans[int ID]
```

Methods

Delete

Deletes an ExecutionPlan [p. 91] object from the ExecutionPlans [p. 96] collection by the specified ID.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlans`

Syntax

```
Delete(int EPDefnId)
```

Exists

Checks for the existence of an ExecutionPlan [p. 91] in the ExecutionPlans [p. 96] collection by the specified name.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlans`

Syntax

```
Exists(string EPName)
```

Load

Fills the ExecutionPlans [p. 96] collection with ExecutionPlan [p. 91] objects.

Type: `void`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlans`

Syntax

```
Load()
```

Refresh

This method refreshes the item(s) from the database.

Type: `void`

Syntax

```
Refresh()
```

RMNewEP

Creates and returns an ExecutionPlan [p. 91] object in the ExecutionPlans [p. 96] collection for the current RequirementFolder [p. 138].

Type: `ExecutionPlan`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlans`

Syntax

```
RMNewEP(string EPName, bool bCreateGroups)
```

- `string EPName` - the name of the new Execution Plan.
- `bool bCreateGroups` - used to create execution groups based on your requirement hierarchy. If false, the *Execution Plan* will contain a flat list of tests.

RMReplaceEP

Updates and returns an existing ExecutionPlan [p. 91] object in the ExecutionPlans [p. 96] collection by removing all the nodes in it and recreating the hierarchy from the current RequirementFolder [p. 138].

Type: `ExecutionPlan`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlans`

Syntax

```
RMReplaceEP(int EPDefnID, string EPName, bool bCreateGroups)
```

- `int EPDefnID` - the Execution Plan ID.
- `string EPName` - the name of the new Execution Plan.
- `bool bCreateGroups` - used to create execution groups based on your requirement hierarchy. If false, the *Execution Plan* will contain a flat list of tests.

RMUpdateEP

Updates and returns an existing ExecutionPlan [p. 91] object in the ExecutionPlans [p. 96] collection by removing all of the nodes in it and recreating the hierarchy from the current RequirementFolder [p. 138].

Type: `ExecutionPlan`

Namespace: `Compuware.QM.QADirector.SDK.ExecutionPlans`

Syntax

```
RMReplaceEP(int EPDefnID, string EPName, bool bCreateGroups)
```

- `int EPDefnID` - the Execution Plan ID.
- `string EPName` - the name of the new Execution Plan.
- `bool bCreateGroups` - used to create execution groups based on your requirement hierarchy. If false, the *Execution Plan* will contain a flat list of tests.

Group

Returns an (Execution) `Group` object. Inherits from TMExecAsset [p. 188].

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- AssetDefnID [p. 188]
- AssetType [p. 189]
- ApplyRulesTo [p. 100]
- AutomatedScriptCount [p. 100]
- CreatedByUser [p. 100]
- CustomAttributeValues [p. 100]
- DateCreated [p. 100]
- DateModified [p. 100]
- Description [p. 101]
- DisplayID [p. 189]
- EstimatedTime [p. 101]
- GroupCount [p. 101]
- ID [p. 101]
- ManualScriptCount [p. 101]
- Mode [p. 101]
- ModifiedByUser [p. 101]
- Name [p. 189]
- RunInParallel [p. 102]

- TestCount [p. 102]

Properties

ApplyRulesTo

Returns the machine name that applies the rules to this Group [p. 99].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Group`

AssetDefnID

Gets the asset ID value of the TMAsset [p. 187].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

AssetType

Returns the type of asset for the TMAsset [p. 187].

Type: `Compuware.QACenter.TM.eAssetTypes`. See eAssetTypes [p. 88].

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

AutomatedScriptCount

Returns the number of Automated Scripts associated to this Group [p. 99].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Group`

CreatedByUser

Returns the `User` that created the Group [p. 99].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Group`

CustomAttributeValues

Returns the CustomAttributeValues [p. 74] collection assigned to the Group [p. 99].

Type: `Compuware.QM.QADirector.SDK.CustomAttributeValues`

Namespace: `Compuware.QM.QADirector.SDK.Group`

DateCreated

Returns the date that the Group [p. 99] was created.

Type: `System.DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Group`

DateModified

Returns the date that the Group [p. 99] was last modified.

Type: `System.DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Group`

Description

Returns the description field of the Group [p. 99].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Group`

DisplayID

Gets the `DisplayID` field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

EstimatedTime

Returns the time estimate in hours for the Group [p. 99].

Type: `decimal`

Namespace: `Compuware.QM.QADirector.SDK.Group`

GroupCount

Returns the number of `Group` objects associated with the current Group [p. 99].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Group`

ID

Returns the ID field of the Group [p. 99].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Group`

ManualScriptCount

Returns the number of ManualScript [p. 112] objects associated with this Group [p. 99].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Group`

Mode

Returns the Group [p. 99] mode.

Type: `Compuware.QACenter.TM.ExecGroupMode`. See ExecGroupMode [p. 84].

Namespace: `Compuware.QM.QADirector.SDK.Group`

ModifiedByUser

Returns the User [p. 192] object that last modified the Group [p. 99].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Group`

Name

Gets the `Name` field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

RunInParallel

Returns whether child groups and tests run at the same time as this group (ChildTestsOnly / ChildGroupsOnly / ChildGroupsAndTests / None) to this `Group`.

Type: `Compuware.QACenter.TM.eExecGroupRunInParallel` See eExecGroupRunInParallel [p. 84].

Namespace: `Compuware.QM.QADirector.SDK.Group`

TestCount

Returns the number of Test [p. 174] objects associated with the Group [p. 99].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Group`

Job

Returns a `Job` object.

See Job.ScheduleOptions [p. 105] for information on how to schedule a job.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- AssignedTo [p. 103]
- CreatedByUser [p. 103]
- Description [p. 103]
- ExecuteCleanupRules [p. 103]
- ExecuteScripts [p. 103]
- ExecuteSetupRules [p. 103]
- ExecutionCycle [p. 104]
- ExecutionMachines [p. 104]
- JobType [p. 104]
- MinimizeWhileRunning [p. 104]
- ModifiedByUser [p. 104]
- Name [p. 104]
- OverrideTimeout [p. 104]

- ResultFolder [p. 154]
- ResultPropagation [p. 105]
- Schedule [p. 105]
- SequentialManualTestExecution [p. 111]
- StopOnFirstScriptFailure [p. 111]
- UseDefaultEnvironementVariables [p. 112]

Methods

- SubmitJob [p. 112]

Properties

AssignedTo

For `Manual Tests`, `AssignedTo` gets or sets the User [p. 192] to which the Job [p. 102] is assigned.

Set `AssignedTo` to `null` to set the job as `Unassigned`.

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Job`

CreatedByUser

Returns the User [p. 192] that created the Job [p. 102].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Job`

Description

Gets or sets the description field for the Job [p. 102].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Job`

ExecuteCleanupRules

If set to true, any Cleanup rules that are set will be executed for the Job [p. 102].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Job`

ExecuteScripts

If set to true, any Scripts in the Job [p. 102] will be executed.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Job`

ExecuteSetupRules

If set to true, any Setup rules that are set will be executed for the Job [p. 102].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Job`

ExecutionCycle

Gets or sets the Execution Cycle (CycleLabel [p. 76]) to use for this Job [p. 102].

Set `ExecutionCycle` to `null` if no cycle should be used.

Type: `Compuware.QM.QADirector.SDK.CycleLabel`

Namespace: `Compuware.QM.QADirector.SDK.Job`

ExecutionMachines

For `Automated Jobs`, gets or sets the list of machines to run a Job [p. 102]. For multiple machines, separate using a comma. For example:

```
machine1,machine2
```

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Job`

JobType

Gets or sets the Job [p. 102] type.

Type: `Compuware.QACenter.TM.eExecType`. See eExecType [p. 84].

Namespace: `Compuware.QM.QADirector.SDK.Job`

MinimizeWhileRunning

Gets or sets a `bool` indicating whether or not *QA*Director will be minimized while the Job [p. 102] is running.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Job`

ModifiedByUser

Returns the User [p. 192] that last modified the Job [p. 102].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Job`

Name

Gets or sets the name field for the Job [p. 102].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Job`

OverrideTimeout

Gets or sets the override timeout value for the Job [p. 102]. It will override the timeout values set in tests.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Job`

ResultFolder

Returns a `ResultFolder` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- CreatedByUser [p. 154]
- Description [p. 155]
- ID [p. 155]
- IsPublic [p. 155]
- ModifiedByUser [p. 155]
- Name [p. 155]
- Results [p. 155]

ResultPropagation

Gets or sets the result propagation value for the Job [p. 102]. If set to true, results will be propagated to related requirements.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Job`

Schedule

Gets the `ScheduleOptions` object that is used to keep track of the schedule options for the Job [p. 102], including Start time and recurring options.

You cannot create a `ScheduleOptions` object, instead, you access all of the scheduling functionality contained within the `ScheduleOption` object. For example:

```
job.Schedule.StartNow = true;
```

Refer to Job.ScheduleOptions [p. 105] to explore all of the different scheduling options for a Job.

Type: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

Namespace: `Compuware.QM.QADirector.SDK.Job`

ScheduleOptions
Job.ScheduleOptions

The `Job.ScheduleOptions` functionality allows you schedule a job to run exactly when and how often you want it to run. These options correspond to the **Schedule** page of the **Job Description** dialog box in *QA*Director. There are three factors to consider when starting a job:

- When will the job start? This corresponds to the **Schedule Time** group in *QA*Director. For more information, see Setting the Schedule Time for a Job [p. 106].
- How will the job recur? This corresponds to the **Recurrence Pattern** group in *QA*Director. For more information, see Setting the Recurrence for a Job [p. 106].

- When will the job end? This corresponds to the **Range of Recurrence** group in *QA*Director. For more information, see Setting the Range of Recurrence for a Job [p. 107].

Schedule Members

- EndsAfterOccurrence [p. 108]
- EndsBy [p. 108]
- RangeOfRecurrence [p. 108]
- RecurrenceDaily [p. 108]
- RecurrenceTime [p. 108]
- RecurrenceWeekly [p. 109]
- RecurrenceYearly [p. 109]
- ScheduleType [p. 110]
- StartNow [p. 111]
- StartTime [p. 111]

Enumerations

- eDailyOptions [p. 85]
- eMonthlyOptions [p. 85]
- eOrdinals [p. 86]
- eSchedRangeRecur [p. 86]
- eScheduleTypes [p. 86]
- eTimeOptions [p. 86]
- eYearlyOptions [p. 87]
- Months [p. 87]

Setting the Schedule Time for a Job

Use the StartNow [p. 111] or StartTime [p. 111] properties of a `Schedule` to control when the job will start. If you elect to set `StartNow` = `true`, then `StartTime` will be ignored.

StartNow Example

```
job.Schedule.StartNow = true;
```

StartTime Example

```
job.Schedule.StartNow = false;
job.Schedule.StartTime = DateTime.Now;
```

Setting the Recurrence for a Job

In order to set a recurrence pattern for a job, you need to use the ScheduleType [p. 110] property to select how the recurrence should happen. The code samples below contain examples for each type of job recurrence.

Time Frequency Example

```
job.Schedule.ScheduleType = eScheduleTypes.TimeFrequency;

//Creates a recurrence schedule that runs every 5 hours
job.Schedule.RecurrenceTime.Option = Job.ScheduleOptions.eTimeOptions.EveryXHour;
job.Schedule.RecurrenceTime.Interval = 5;
```

Daily Example

```
job.Schedule.ScheduleType = eScheduleTypes.Daily;

//Runs every 3 days
job.Schedule.RecurrenceDaily.Option = Job.ScheduleOptions.eDailyOptions.EveryXDay;
job.Schedule.RecurrenceDaily.Interval = 3;
```

Weekly Example

```
job.Schedule.ScheduleType = eScheduleTypes.Weekly;

//Runs every 2 weeks on Saturday and Sunday
job.Schedule.RecurrenceWeekly.Option.Saturday = true;
job.Schedule.RecurrenceWeekly.Option.Sunday = true;
job.Schedule.RecurrenceWeekly.Interval = 2;
```

Monthy Examples

```
job.Schedule.ScheduleType = eScheduleTypes.Monthly;

//Runs every second monday every 6 months
job.Schedule.RecurrenceMonthly.Option =
Job.ScheduleOptions.eMonthlyOptions.OrdinalDayofWeek;
job.Schedule.RecurrenceMonthly.AdditionalOptions.OrdinalOption =
Job.ScheduleOptions.eOrdinals.Second;
job.Schedule.RecurrenceMonthly.AdditionalOptions.DayOfTheWeek = DayOfWeek.Monday;
job.Schedule.RecurrenceMonthly.Interval = 6;

//Runs the fifth day of every 2 months
job.Schedule.RecurrenceMonthly.Option = Job.ScheduleOptions.eMonthlyOptions.SpecificDay;
job.Schedule.RecurrenceMonthly.AdditionalOptions.DayOfMonth = 5;
job.Schedule.RecurrenceMonthly.Interval = 2;
```

Yearly Examples

```
job.Schedule.ScheduleType = eScheduleTypes.Yearly;

//Creates a recurring job that runs every March 15th
job.Schedule.RecurrenceYearly.Option = Job.ScheduleOptions.eYearlyOptions.MonthOption;
job.Schedule.RecurrenceYearly.AdditionalOptions.Month = Job.ScheduleOptions.Months.March;
job.Schedule.RecurrenceYearly.Interval = 15;

//Creates a recurring job that runs every 2nd Thursday of the year
job.Schedule.RecurrenceYearly.Option = Job.ScheduleOptions.eYearlyOptions.DayofWeekOption;
job.Schedule.RecurrenceYearly.AdditionalOptions.OrdinalOption =
Job.ScheduleOptions.eOrdinals.Second;
job.Schedule.RecurrenceYearly.AdditionalOptions.DayOfTheWeek = DayOfWeek.Thursday;
```

Setting the Range of Recurrence for a Job

After selecting when a job starts and how it should recur, you need to set when it will end. Use the RangeOfRecurrence [p. 108] property to specifiy when a job ends.

eSchedRangeRecur.NumberofOccur

```
//Set the range of recurrence to only run once (this is also the default).
job.Schedule.RangeOfRecurrence = eSchedRangeRecur.NumberofOccur;
job.Schedule.EndsAfterOccurrence = 1;
```

eSchedRangeRecur.EndDate

```
//Set the range of recurrence to end by a specified date.
job.Schedule.RangeOfRecurrence = eSchedRangeRecur.EndDate;
DateTime FinishDate = DateTime.Now.AddDays(14);
job.Schedule.EndsBy = FinishDate;
```

eSchedRangeRecur.NoEndDate

```
job.Schedule.RangeOfRecurrence = eSchedRangeRecur.NoEndDate;
```

Members

EndsAfterOccurrence

If the RangeOfRecurrence [p. 108] is set to `NumberofOccur`, use this to get or set the number of times a job will recurr.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

EndsBy

Specifies the date on which the job will end. Used by the RangeOfRecurrence [p. 108]property.

Type: `DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

RangeOfRecurrence

This member of Job.ScheduleOptions [p. 105] gets or sets the type of range of recurrence:

- No end date
- A specific end date
- After a number of ocurrences

Type: `enum`. See eSchedRangeRecur [p. 86].

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

Code Sample

See Setting the Schedule Time for a Job [p. 106].

RecurrenceDaily

If Schedule [p. 105] is set to `Daily`, use this property to get or set recurrence patterns based on days.

Type: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions.RecurOptions`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

RecurrenceMonthly

If Schedule [p. 105] is set to `Monthly`, use this property to get or set recurrence patterns based on months.

Type: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions.RecurOptions`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

RecurrenceTime

If Schedule [p. 105] is set to `TimeFrequency`, use this property to get or set recurrence patterns based on time.

Type: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions.RecurOptions`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

RecurrenceWeekly

If Schedule [p. 105] is set to `Weekly`, use this property to get or set recurrence patterns based on weeks.

Type: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions.RecurOptions`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

RecurrenceYearly

If Schedule [p. 105] is set to `Yearly`, use this property to get or set recurrence patterns based on years.

Type: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions.RecurOptions`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

Recurrence Type Members

Each recurrence type contains the following members to help define when a job should run:

AdditionalOptions

`AdditionalOptions` contain further refinements to the Option member for monthly and yearly recurrence types.

Interval

Interval represents the numeric value for any recurrence type.

Option

`Option` is used to select which option or options should be used depending on the recurrence type. For instance, Time recurrence types can be set to either minutes or hours.

The following table contains the available values for each member.

Recurrence Types	`AdditionalOptions`	`Interval`	`Option`
RecurrenceTime [p. 108]	NA	Use this to set either the number of minutes or hours depending on the `Option` set.	eTimeOptions [p. 86]
RecurrenceDaily [p. 108]	NA	Use this to set either the number of days or weeks depending on the `Option` set.	eDailyOptions [p. 85]
RecurrenceWeekly [p. 109]	NA	Use this to set the number of weeks to run the job depending on the `Option` set. Example: Every <`Interval`> weeks on <`Option`>	Weekly options include the days of the week: • `Sunday` • `Monday` • `Tuesday` • `Wednesday` • `Thursday`

Recurrence Types	AdditionalOptions	Interval	Option
			• Friday • Saturday
RecurrenceMonthly [p. 108]	• DayOfMonth - Use with the SpecificDay option. Represents the day of the month to run the job. Numbers should range from 1 to 30. Use 0 in order to run the job on the last day of the month. Numbers outside the allowed range will be set to 0. • DayOfTheWeek - Used with the OrdinalDayofWeek Option to indicate the day of the week. • OrdinalOption - Used with the OrdinalDayofWeek Option to indicate the ordinal day of the week. See eOrdinals [p. 86].	Use this to set the number of months to run the job. Example: Runs x days every <Interval> months	eMonthlyOptions [p. 85]
RecurrenceYearly [p. 109]	• DayOfTheWeek - Used with the DayofWeekOption Option to indicate the day of the week. • Month - Use with MonthOption to indicate which month the recurring schedule runs on. See Months [p. 87]. • OrdinalOption - Used with the DayofWeekOption Option to indicate the ordinal day of the week. See eOrdinals [p. 86].	Use with the Month option to set the day of the month to run the job.	eYearlyOptions [p. 87]

ScheduleType

Gets or sets the recurrence pattern for recurring jobs. Depending on the setting, one of the following recurrence pattern properties should be set:

- RecurrenceTime [p. 108]
- RecurrenceDaily [p. 108]
- RecurrenceWeekly [p. 109]
- RecurrenceMonthly [p. 108]
- RecurrenceYearly [p. 109]

Type: Compuware.QACenter.TM.eScheduleType. See eScheduleTypes [p. 86].

Namespace: Compuware.QM.QADirector.SDK.Job.ScheduleOptions

Depending on the `ScheduleType` selected, each recurrence pattern contains additional options to set. For more information, see Recurrence Type Members [p. 109].

StartNow

This member of Job.ScheduleOptions [p. 105] gets or sets a `bool` indicating whether or not the Job [p. 102] starts when SubmitJob [p. 112] is called.

NOTE

if `StartNow` = `true`, it will override the `StartTime` property.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

Code Sample

See Setting the Schedule Time for a Job [p. 106].

StartTime

This member of Job.ScheduleOptions [p. 105] gets or sets a `DateTime` value indicating when the Job [p. 102] will start.

NOTE

if `StartNow` = `true`, it will override the `StartTime` property.

Type: `DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Job.ScheduleOptions`

Code Sample

See Setting the Schedule Time for a Job [p. 106].

SequentialManualTestExecution

Gets or sets the `SequentialManualTestExecution` field for the Job [p. 102]. For `Manual Test` Jobs, if this is set to `true`, during Execution the order of Script Execution will only be allowed to run in the order they are listed in the Execution Plan. This is the **Test Execution Order** option in the **Job Description** dialog box.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Job`

StopOnFirstScriptFailure

Gets or sets the `StopOnFirstScriptFailure` field for the Job [p. 102]. If set to `true`, the Job will stop running after the first Script fails.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Job`

UseDefaultEnvironementVariables

Gets or sets the UseDefaultEnvironmentVariables field for the Job [p. 102]. If set to `true` during `Automatted job` execution, the default Microsoft Windows environment variables will be set for any running applications.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Job`

Methods

SubmitJob

Submits the current Job [p. 102] for execution.

NOTE

`SubmitJob` returns `false` if it fails. Call `GetLastError` to return the error string.

Type: `bool`.

Namespace: `Compuware.QM.QADirector.SDK.Job`

Syntax

```
SubmitJob()
```

ManualScript

Returns a `ManualScript` object. Inherits from Scripts [p. 172].

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- AssetDefnID [p. 188]
- AssetType [p. 189]
- CreatedByUser [p. 113]
- Description [p. 189]
- DisplayID [p. 189]
- ManualSteps [p. 113]
- ModifiedByUser [p. 113]
- Name [p. 189]

Properties

AssetDefnID

Gets the asset ID value of the TMAsset [p. 187].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

AssetType

Returns the type of asset for the TMAsset [p. 187].

Type: `Compuware.QACenter.TM.eAssetTypes`. See eAssetTypes [p. 88].

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

CreatedByUser

Returns the User [p. 192] that created the ManualScript [p. 112].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.ManualScript`

Description

Gets the description field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

DisplayID

Gets the `DisplayID` field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

ManualSteps

Returns the ManualSteps [p. 118] collection assigned to the ManualScript [p. 112].

Type: `Compuware.QM.QADirector.SDK.ManualSteps`

Namespace: `Compuware.QM.QADirector.SDK.ManualScript`

ModifiedByUser

Returns the User [p. 192] that last modified the ManualScript [p. 112].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.ManualScript`

Name

Gets the `Name` field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

ManualStep

Returns a `ManualStep` object.

The *QA*Director API enables two ways to create `ManualSteps`.

1. Use the `ManualSteps.New` method to create a new `ManualStep` and add it directly to the database.

2. Create a new `ManualStep` of the specified type (ManualStep.Step [p. 116]) and then use the `ManualSteps.Add()` and `ManualSteps.Update()` methods. For more information, see ManualSteps.Add Code Sample [p. 118].

NOTE

Use the Update [p. 116] method to save changes to a `ManualStep`.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- AssociatedData [p. 114]
- Attachment [p. 114]
- CreatedByUser [p. 114]
- ExpectedAnswer [p. 115]
- FileName [p. 115]
- ModifiedByUser [p. 115]
- Notes [p. 115]
- PossibleAnswer [p. 115]
- StepNo [p. 115]
- StepText [p. 115]
- StepType [p. 115]
- StepTypeAsString [p. 116]

Methods

Update [p. 116]

Properties

AssociatedData

Returns the associated data field of the ManualStep [p. 113].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

Attachment

Returns the attachment field of the ManualStep [p. 113] as a byte array.

Type: `byte[]`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

CreatedByUser

Returns the User [p. 192] that created the ManualStep [p. 113].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

ExpectedAnswer

Returns the expected answer field of the ManualStep [p. 113].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

FileName

Returns the file name field of the ManualStep [p. 113].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

ModifiedByUser

Returns the User [p. 192] that last modified the ManualStep [p. 113].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

Notes

Returns the notes field of the ManualStep [p. 113].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

PossibleAnswer

Returns the possible answer field of the ManualStep [p. 113].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

StepNo

Returns the step number field of the ManualStep [p. 113].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

StepText

Returns the step text field of the ManualStep [p. 113].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

StepType

Returns the step type field as an `int` of the ManualStep [p. 113].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

StepTypeAsString

Returns the step type field of the ManualStep [p. 113].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

Methods

Update

Updates the database with the changes made to already existing steps of the ManualStep [p. 113].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

Syntax

`Update()`

`Update(bool bIsScriptDirty)` - `bIsScriptDirty` indicates if the Script has changes.

Creating Manual Step Types

ManualStep.Step

Use the `ManualSteps.Add(ManualStep.Step)` method to create manual steps of specific types. You can create `ManualSteps` of the types listed below. Then, use the `Add` method to add them and the `Update` method to save them to the database.

- Instruction [p. 116]
- MultipleChoice [p. 117]
- PassFail [p. 117]
- TrueFalse [p. 117]
- YesNo [p. 118]

Manual Step Enums

- PassFail.CorrectAnswer [p. 87]
- TrueFalse.CorrectAnswer [p. 88]
- YesNo.CorrectAnswer [p. 88]

Instruction

Creates a new `ManualStep` of type `Instruction`.

Type: `Compuware.QM.QADirector.SDK.ManualStep.Instruction`

Namespace: `Compuware.QM.QADirector.SDK`

Syntax

- **`Instruction(string text, string description)`**
- **`Instruction(string text, string description, string notes, string assocdata, string url)`**
- **`Instruction(string text, string description, string notes, string assocdata, string filename, byte[] attachment)`**

MultipleChoice

Creates a new `ManualStep` of type `MultipleChoice`.

Type: `Compuware.QM.QADirector.SDK.ManualStep.MultipleChoice`

Namespace: `Compuware.QM.QADirector.SDK`

Syntax

- **`MultipleChoice(string text, string description, string[] possibleanswers, int expectedanswerindex, string notes, string assocdata, string url)`**
- **`MultipleChoice(string text, string description, string[] possibleanswers, int expectedanswerindex)`**
- **`MultipleChoice(string text, string description, string[] possibleanswers, int expectedanswerindex, string notes, string assocdata, string filename, byte[] attachment)`**

PassFail

Creates a new `ManualStep` of type `PassFail`.

Type: `Compuware.QM.QADirector.SDK.ManualStep.PassFail`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

Syntax

- **`PassFail(string text, string description, CorrectAnswer answer, string notes, string assocdata, string url)`**
- **`PassFail(string text, string description, CorrectAnswer answer, string notes, string assocdata, string filename, byte[] attachment)`**
- **`PassFail(string text, string description)`**

TrueFalse

Creates a new `ManualStep` of type `TrueFalse`.

Type: `Compuware.QM.QADirector.SDK.ManualStep.TrueFalse`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

Syntax

- **`TrueFalse(string text, string description)`**
- **`TrueFalse(string text, string description, CorrectAnswer answer, string notes, string assocdata, string url)`**

- **`TrueFalse(string text, string description, CorrectAnswer answer, string notes, string assocdata, string filename, byte[] attachment)`**

YesNo

Creates a new `ManualStep` of type `YesNo`.

Type: `Compuware.QM.QADirector.SDK.ManualStep.YesNo`

Namespace: `Compuware.QM.QADirector.SDK.ManualStep`

Syntax

- **`YesNo(string text, string description, CorrectAnswer answer, string notes, string assocdata, string url)`**
- **`YesNo(string text, string description, CorrectAnswer answer, string notes, string assocdata, string filename, byte[] attachment)`**
- **`YesNo(string text, string description)`**

ManualSteps.Add Code Sample

```
int id = (int)idN;

//get the script object
ManualScript srpt = (ManualScript)curFolder.Scripts[id];

//create steps with required steptypes
ManualStep.TrueFalse TF = new ManualStep.TrueFalse("step2",
"desc2", ManualStep.TrueFalse.CorrectAnswer.False, "", "", "");

ManualStep.Instruction instruction = new ManualStep.Instruction("step1",
"desc1", "notes test", "assoc data test","http://www.compuware.com");

ManualStep.PassFail PF = new ManualStep.PassFail("step2", "desc2",
ManualStep.PassFail.CorrectAnswer.Pass, "", "", "");

string[] choices = new string[4];
choices[0] = "a";
choices[1] = "b";
choices[2] = "c";
choices[3] = "d";

System.IO.FileStream fs = new System.IO.FileStream("C:\\MTattachmenttest.txt",
System.IO.FileMode.Open, System.IO.FileAccess.Read);
Byte[] blob = new Byte[fs.Length];
fs.Read(blob, 0, blob.Length);
fs.Close();

ManualStep.MultipleChoice MC = new ManualStep.MultipleChoice("step3",
"desc3", choices, 1, "mcnotes", "mcassdata", "sample.txt", blob);

//add the steps to Manualsteps collection
srpt.ManualSteps.Add(instruction);
srpt.ManualSteps.Add(TF);
srpt.ManualSteps.Add(PF);
srpt.ManualSteps.Add(MC);

//do an update of manual steps
bool bUpdated = srpt.ManualSteps.Update();
```

ManualSteps

Returns a collection of `ManualStep` objects.

The *QA*Director API enables two ways to create `ManualSteps`.

1. Use the `ManualSteps.New` method to create a new `ManualStep` and add it directly to the database.
2. Create a new `ManualStep` of the specified type (ManualStep.Step [p. 116]) and then use the `ManualSteps.Add()` and `ManualSteps.Update()` methods. For more information, see ManualSteps.Add Code Sample [p. 118].

NOTE

The `New` method creates a new `ManualStep` and adds it directly to the database. The `Add` method creates a new `ManualStep` of the `ManualStep.Step` type. You must use the `Update()` method to update the database after modifying items in the collection with `Add`.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 119]
- ManualSteps [p. 119]

Methods

- Add [p. 120]
- DeleteAll [p. 120]
- New [p. 120]
- Refresh [p. 196]
- RemoveManualStep [p. 121]
- Update [p. 121]

Classes

- MTEDefaultStepTypes [p. 121]

Properties

Count

Gets the number of ManualStep [p. 113] objects in the ManualSteps [p. 118] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ManualSteps`

ManualSteps

`ManualSteps` indexer that returns a ManualStep [p. 113] object.

Type: `Compuware.QM.QADirector.SDK.ManualStep`

Namespace: `Compuware.QM.QADirector.SDK.ManualSteps`

Syntax

```
ManualSteps[int ID]
```

```
ManualSteps[String Name]
```

Methods

Add

Adds a new ManualStep [p. 113] object to the ManualSteps [p. 118] collection. Use the Update [p. 121] method to update the database.

Type: `void`

Namespace: `Compuware.QM.QADirector.SDK.ManualSteps`

Syntax

```
Add(ManualStep.Step step)
```

DeleteAll

Deletes all ManualStep [p. 113] objects in the ManualSteps [p. 118] collection. Use the Update [p. 121] method to update the database.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.ManualSteps`

Syntax

```
DeleteAll()
```

New

Creates and adds a new ManualStep [p. 113] object in the `ManualSteps` collection with the specified parameters.

Type: `Compuware.QM.QADirector.SDK.ManualStep`

Namespace: `Compuware.QM.QADirector.SDK.ManualSteps`

Syntax

- New(string ScriptName, int OrderNumber, string StepDesc,int MTEDefaultStepTypes [p. 121], string PossibleAnswers, string ExpectedAnswer, string Notes, string AssociatedData, string Attachment, string AttachmentFileNameWithExtension)
- New(string ScriptName, int OrderNumber, string StepDesc, int MTEDefaultStepTypes [p. 121], string PossibleAnswers, string ExpectedAnswer, string Notes, string AssociatedData, byte[] Attachment, string AttachmentFileNameWithExtension)
- New(int OrderNumber, string StepDesc, int MTEDefaultStepTypes [p. 121], string PossibleAnswers, string ExpectedAnswer, string Notes, string AssociatedData, string Attachment, string AttachmentFileNameWithExtension)

- New(int OrderNumber, string StepDesc, int MTEDefaultStepTypes [p. 121], string PossibleAnswers, string ExpectedAnswer, string Notes, string AssociatedData, byte[] Attachment, string AttachmentFileNameWithExtension)
- New(int OrderNumber, string StepDesc, int MTEDefaultStepTypes [p. 121], string PossibleAnswers, string ExpectedAnswer, string Notes, string AssociatedData, bool bIsAttachmentFile, string AttachmentFilePath, string AttachmentFileNameWithExtension)

Refresh

This method refreshes the item(s) from the database.

Type: `void`

Syntax

```
Refresh()
```

RemoveManualStep

Removes the specified ManualStep [p. 113] object in the ManualSteps [p. 118] collection with the specified step number.

Use the Update [p. 121] method to update the database.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.ManualSteps`

Syntax

```
RemoveManualStep(int iStepNo)
```

Update

Updates all of the ManualStep [p. 113] objects in the ManualSteps [p. 118] collection (add/remove steps) and saves into the database.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.ManualSteps`

Syntax

```
Update()
Update(bool bIsScriptDirty)
```

Classes

MTEDefaultStepTypes

Provides a list of possible Step Types when using the `ManualSteps.New` method. See New [p. 120].

Values:

- `QUESTIONTEXT = 1`

- `MULTIPLECHOICE = 2`
- `INSTRUCTION = 3`
- `QUESTION = 7`

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK`

Project

Provides access to all the information for a given project in *QA*Director.

NOTE

Use the OpenProject [p. 128] method to open a project once you obtain a reference.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- CreatedByUser [p. 123]
- CustomAttributes [p. 123]
- CustomAttributeValues [p. 123]
- CycleLabels [p. 123]
- CycleName [p. 123]
- Description [p. 124]
- EndDate [p. 124]
- ExecutionPlans [p. 124]
- ID [p. 124]
- MaxRiskWeight [p. 124]
- ModifiedByUser [p. 124]
- Name [p. 125]
- NumberOfCycles [p. 125]
- ProjectDefects [p. 125]
- ProjectUsers [p. 125]
- RequirementFolders [p. 125]
- ResultFolders [p. 125]
- RiskModel [p. 125]
- RiskModelID [p. 125]
- RMFolder [p. 126]
- Scripts [p. 126]
- StartDate [p. 126]

- TestFolders [p. 126]
- Tests [p. 126]
- UseScriptsTimeEst [p. 126]

Methods

- AddScript [p. 127]
- AddUserToProject [p. 127]
- CloseProject [p. 127]
- GetResult [p. 127]
- GetScript [p. 128]
- OpenProject [p. 128]
- ProjectCAExists [p. 128]
- UpdateCAs [p. 128]

Properties

CreatedByUser

Returns the User [p. 192] that created the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Project`

CustomAttributes

Returns the CustomAttributes [p. 71] collection assigned to the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.CustomAttributes`

Namespace: `Compuware.QM.QADirector.SDK.Project`

CustomAttributeValues

Returns the CustomAttributeValues [p. 74] collection for the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.CustomAttributeValues`

Namespace: `Compuware.QM.QADirector.SDK.Project`

CycleLabels

Returns the CyclesLabels [p. 78] collection associated with the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.CyclesLabels`

Namespace: `Compuware.QM.QADirector.SDK.Project`

CycleName

Returns the `CycleName` for the Project [p. 122].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Project`

Description

Gets/sets the Project [p. 122] description.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Project`

EndDate

Gets the end date of the Project [p. 122].

Type: `System.DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Project`

ExecutionPlans

Gets the ExecutionPlans [p. 96] collection associated with the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.ExecutionPlans`

Namespace: `Compuware.QM.QADirector.SDK.Project`

GetScript

Returns the specified Script [p. 164] object from the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.Script`

Namespace: `Compuware.QM.QADirector.SDK.Project`

Syntax

```
GetScript(int ScriptDefnID)
```

Parameters

`int ScriptDefnID` is the ID of the Script to return.

ID

Gets the ID of the Project [p. 122].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Project`

MaxRiskWeight

Gets the `MaxRiskWeight` field associated with the `Project`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Project`

ModifiedByUser

Returns the User [p. 192] that last modified the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Project`

Name

Gets the Project [p. 122] name.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Project`

NumberOfCycles

Gets/sets the number of cycles associated with the Project [p. 122].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Project`

ProjectDefects

Gets the Defects [p. 81] collection associated with the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.Defects`

Namespace: `Compuware.QM.QADirector.SDK.Project`

ProjectUsers

Gets the Users [p. 195] collection associated with the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.Users`

Namespace: `Compuware.QM.QADirector.SDK.Project`

RequirementFolders

Gets the RequirementFolders [p. 145] collection associated with the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.RequirementFolders`

Namespace: `Compuware.QM.QADirector.SDK.Project`

ResultFolders

Gets the collection of ResultFolders [p. 155] collection associated with the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.ResultFolders`

Namespace: `Compuware.QM.QADirector.SDK.Project`

RiskModel

Gets the ProjectRiskModel [p. 128] object associated with the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.ProjectRiskModel`

Namespace: `Compuware.QM.QADirector.SDK.Project`

RiskModelID

Gets the `RskModelID` field associated with the Project [p. 122].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Project`

RMFolder

Gets the RMFolder [p. 209] associated with the Project [p. 122].

NOTE

RMFolder should only be used when integrating QADirector with a requirements management solution. See Requirements Management Classes [p. 197].

Type: `Compuware.QM.QADirector.SDK.RMFolder`

Namespace: `Compuware.QM.QADirector.SDK.Project`

Scripts

Gets the Scripts [p. 172] object associated with the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.Scripts`

Namespace: `Compuware.QM.QADirector.SDK.Project`

StartDate

Gets the Start date of the Project [p. 122].

Type: `System.DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Project`

TestFolders

Gets the TestFolders [p. 183] colection object associated with the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.TestFolders`

Namespace: `Compuware.QM.QADirector.SDK.Project`

Tests

Gets all of the tests associated with the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.Tests`

Namespace: `Compuware.QM.QADirector.SDK.Project`

UseScriptsTimeEst

Gets the `UseScriptsTimeEst` field for the Project [p. 122]. This field correlates to the **Time Estimate** section of the **Project Properties** dialog box in *QA*Director. In that area, you have the option of selecting **Tests** or **Scripts** for calculations.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Project`

Methods

AddScript

Adds a global script to the Project [p. 122].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Project`

Syntax

```
AddScript(Script srpt,int iFolderID)
```

Script srpt
The Script [p. 164] object.

int iFolderID
The Id of the folder/tool domain to which the script belongs.

AddUserToProject

Adds a specified `User` to the Project [p. 122] with a specified `Role`.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Project`

Syntax

```
AddUserToProject(User user, Role ProjectRole)
```

- `User` is a User [p. 192] object.
- `Role` is a Role [p. 162] object.

CloseProject

Closes a Project [p. 122].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Project`

Syntax

```
CloseProject()
```

GetResult

Returns the specified Result [p. 159] object for the Project [p. 122] with the specified ID.

Type: `Compuware.QM.QADirector.SDK.Result`

Namespace: `Compuware.QM.QADirector.SDK.Project`

Syntax

```
GetResult(int ResultID)
```

GetScript

Returns the specified Script [p. 164] object from the Project [p. 122].

Type: `Compuware.QM.QADirector.SDK.Script`

Namespace: `Compuware.QM.QADirector.SDK.Project`

Syntax

```
GetScript(int ScriptDefnID)
```

Parameters

`int ScriptDefnID` is the ID of the Script to return.

OpenProject

Opens a Project [p. 122].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Project`

Syntax

```
OpenProject()
```

ProjectCAExists

Returns a bool representing whether or not a `CustomAttribute` with the specified ID exists in the Project [p. 122].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Project`

Syntax

```
ProjectCAExists(int CAID)
```

UpdateCAs

Updates the Custom Attributes belonging to the Project [p. 122].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Project`

Syntax

```
UpdateCAs()
```

ProjectRiskModel

Returns a `ProjectRiskModel` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

Methods

Properties

Attributes

Gets the CustomAttributes [p. 71] collection associated with the ProjectRiskModel [p. 128].

Type: `Compuware.QM.QADirector.SDK.CustomAttributes`

Namespace: `Compuware.QM.QADirector.SDK.ProjectRiskModel`

CreatedByUser

Returns the User [p. 192] that created the ProjectRiskModel [p. 128].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.ProjectRiskModel`

Description

Gets the description field for the ProjectRiskModel [p. 128].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ProjectRiskModel`

ModifiedByUser

Returns the User [p. 192] that last modified the ProjectRiskModel [p. 128].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.ProjectRiskModel`

Name

Gets the name of the ProjectRiskModel [p. 128].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ProjectRiskModel`

RiskModelID

Gets the ID of the ProjectRiskModel [p. 128].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ProjectRiskModel`

Methods

GetRisklLabelFromValue

Gets the risk label string associated with the ProjectRiskModel [p. 128] with the specified numeric risk value.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ProjectRiskModel`

Syntax

```
GetRisklLabelFromValue(int riskvalue)
```

GetRiskType

Gets the risk type `int` associated with the ProjectRiskModel [p. 128] with the specified numeric risk association.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ProjectRiskModel`

Syntax

```
GetRiskType(int riskAssociation)
```

GetRiskValueFromLabel

Gets the risk value `int` associated with the ProjectRiskModel [p. 128] with the specified label.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ProjectRiskModel`

Syntax

```
GetRiskValueFromLabel(string label)
```

Projects

Provides access to all of the projects in a given *QA*Director Client.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 131]
- Projects [p. 131]

Methods

- Exists [p. 131]
- New [p. 131]
- Refresh [p. 132]

Properties

Count

Gets the number of Project [p. 122] objects in the Projects [p. 131] collection for the Client [p. 60].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Projects`

Projects

Syntax

- **`Projects[int projectid]`** - Returns a **`Project`** object by project id via integer.
- **`Projects[string projectname]`** - Returns a **`Project`** object by project name via string.

Methods

Exists

Validates the existence of a Project [p. 122] object in the Projects [p. 131]collection.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Projects`

Syntax

```
Exists(string projectname)
```

Parameters

`String projectname` - the name of the project.

New

Creates a new Project [p. 122] with the given name and description in the current Projects [p. 131] collection.

Type: `Compuware.QM.QADirector.SDK.Project`

Namespace: `Compuware.QM.QADirector.SDK.Projects`

Syntax

```
New(String projectname, String projectdescription)
```

String projectname
Name of the new project.

String projectdescription
Description of the new project.

Refresh

Refreshes the **`Projects`** collection with the latest list of projects from database.

Type: `void`

Namespace: `Compuware.QM.QADirector.SDK.Projects`

Syntax

```
Refresh()
```

Requirement

Returns a `Requirement` object.

Inherits from TMAsset [p. 187].

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- AssetDefnID [p. 188]
- AssetType [p. 189]
- AssignedToUserName [p. 133]
- AssocTests [p. 134]
- AssociatedDefects [p. 134]
- CreatedByUser [p. 134]
- CustomAttributeValues [p. 134]
- DefectCount [p. 134]
- Description [p. 189]
- DisplayID [p. 189]
- EstimatedTime [p. 134]
- ExternalID [p. 135]
- ExternalRMID [p. 135]
- FailedCount [p. 135]
- FolderID [p. 135]
- InProgressCount [p. 135]
- ModifiedByUser [p. 135]
- Name [p. 189]

- NotExecutableCount [p. 135]
- NotStartedCount [p. 136]
- NotSubmittedCount [p. 136]
- ParentReqRelationID [p. 136]
- PassedCount [p. 136]
- ReqRelationID [p. 136]
- RequirementCount [p. 136]
- Risk [p. 136]
- RiskLabels [p. 136]
- ScriptCount [p. 137]
- SiblingOrder [p. 137]
- Status [p. 137]
- TestCount [p. 137]

Methods

- AssociateTest [p. 137]
- CreateTest [p. 137]
- InsertNewRequirement [p. 138]
- SaveRequirementProperties [p. 138]
- UpdateCAs [p. 138]

Properties

AssetDefnID

Gets the asset ID value of the TMAsset [p. 187].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

AssetType

Returns the type of asset for the TMAsset [p. 187].

Type: `Compuware.QACenter.TM.eAssetTypes`. See eAssetTypes [p. 88].

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

AssignedToUserName

Sets or gets the User Name assigned to the Requirement [p. 132].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

AssociatedDefects

Gets the associated Defects [p. 81] collection related to the Requirement [p. 132].

Type: `Compuware.QM.QADirector.SDK.Defects`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

AssocTests

Gets the AssociatedTests [p. 59] collection related to the Requirement [p. 132].

Type: `Compuware.QM.QADirector.SDK.AssociatedTests`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

CreatedByUser

Returns the User [p. 192] that created the Requirement [p. 132].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

CustomAttributeValues

Gets the CustomAttributeValues [p. 74] collection related to the Requirement [p. 132].

Type: `Compuware.QM.QADirector.SDK.CustomAttributeValues`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

DefectCount

Gets the number of defects associated to the Requirement [p. 132].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

Description

Gets the description field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

DisplayID

Gets the `DisplayID` field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

EstimatedTime

Gets the total estimated time of the Requirement [p. 132].

Type: `decimal`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

ExternalID

Returns the external id field of the Requirement [p. 132].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

ExternalRMID

Returns the external RM ID of the Requirement [p. 132].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

FailedCount

Gets the total number of Tests in the Requirement [p. 132] with a status of `Failed`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

FolderID

Returns the parent folder id of the Requirement [p. 132].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

InProgressCount

Returns the total number of Tests in the Requirement [p. 132] with a status of `In Progress`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

ModifiedByUser

Returns the User [p. 192] that last modified the Requirement [p. 132].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

Name

Gets the `Name` field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

NotExecutableCount

Gets the total number of Tests in the Requirement [p. 132] with a status of `Not Executable`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

NotStartedCount

Gets the total number of Tests in the Requirement [p. 132] with a status of `Not Started`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

NotSubmittedCount

Gets the total number of Tests in the Requirement [p. 132] with a status of `Not Submitted`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

ParentReqRelationID

Returns the parent requirement relation ID of the Requirement [p. 132].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

PassedCount

Gets the total number of Tests in the Requirement [p. 132] with a status of `Passed`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

ReqRelationID

Gets the requirement relation ID of the Requirement [p. 132].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

RequirementCount

Gets the total number of child requirements in the Requirement [p. 132].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

Risk

Gets the risk value in a string of the Requirement [p. 132].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

RiskLabels

Returns a string array of the Risk Labels belonging to the current Requirement [p. 132].

Type: `string[]`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

ScriptCount

Gets the total number of Scripts in the Requirement [p. 132].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

SiblingOrder

Gets the sibling order of the Requirement [p. 132] which indicates this requirement's position relative to sibling requirements.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

Status

Gets the status field of the Requirement [p. 132], either:

- `Complete`
- `In Progress`

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

TestCount

Gets the total number of Tests in the Requirement [p. 132].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

Methods

AssociateTest

Associates a Test [p. 174] with the current Requirement [p. 132].

Type: `Compuware.QM.QADirector.SDK.RequirementNode`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

Syntax

```
AssociateTest(RequirementNode reqNode,Test objTest)
```

CreateTest

Creates a Test [p. 174] belonging to the current Requirement [p. 132].

Type: `Compuware.QM.QADirector.SDK.RequirementNode`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

Syntax

```
CreateTest(RequirementNode reqNode)
```

InsertNewRequirement

Inserts a new child requirement belonging to the current Requirement [p. 132].

Type: `Compuware.QM.QADirector.SDK.RequirementNode`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

Syntax

```
InsertNewRequirement(RequirementNode reqNode)
```

SaveRequirementProperties

Saves the properties of the current Requirement [p. 132].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

Syntax

```
SaveRequirementProperties()
```

UpdateCAs

Updates the CustomAttributeValues [p. 74] collection of the the current Requirement [p. 132].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Requirement`

Syntax

```
UpdateCAs()
```

RequirementFolder

Returns a `RequirementFolder` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- CreatedByUser [p. 210]
- Description [p. 210]
- ExecutionPlans [p. 210]
- ID [p. 210]
- IsPublic [p. 210]
- ModifiedByUser [p. 210]
- Name [p. 210]
- Nodes [p. 210]
- RequirementNodes [p. 211]
- Requirements [p. 211]

- RMIntegrationValues [p. 211]
- SetRMUpdateOptsGetRiskValue [p. 212]
- Tests [p. 212]

Methods

- CreateCSVTree [p. 212]
- CreateRMTree [p. 212]
- GetCSVErrorLog [p. 212]
- InsertNewRequirement [p. 213]
- ResetAndDeleteRMIntegration [p. 213]
- ResetRMIntegration [p. 213]
- RMNewEP [p. 214]
- RMReplaceEP [p. 214]
- RMUpdateEP [p. 211]
- UpdateRMTree [p. 214]

Properties

CreatedByUser

Returns the User [p. 192] that created the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Description

Gets the description field for the RequirementFolder [p. 138].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

ExecutionPlans

Gets the ExecutionPlans [p. 96] associated with the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.ExecutionPlans`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

ID

Gets the id value for the RequirementFolder [p. 138].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

IsPublic

Returns a `bool` indicating the availability of the RequirementFolder [p. 138].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

ModifiedByUser

Returns the User [p. 192] that last modified the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Name

Gets the name for the RequirementFolder [p. 138].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Nodes

Gets the Nodes [p. 206] collection for the RequirementFolder [p. 138].

NOTE

This property should only be used if you are performing an integration with a requirements management tool. See Requirements Management Classes [p. 197].

Type: `Compuware.QM.QADirector.SDK.Nodes`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

RequirementNodes

Gets the RequirementNodes [p. 148] collection for the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.RequirementNodes`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Requirements

Gets the Requirements [p. 149] collection for the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.Requirements`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

RMIntegrationValues

Gets the RMIntegrationValues [p. 217] collection for the RequirementFolder [p. 138].

NOTE

This property should only be used if you are performing an integration with a requirements management tool. See Requirements Management Classes [p. 197].

Type: `Compuware.QM.QADirector.SDK.Nodes`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

SetRMUpdateOptsGetRiskValue

Sets a value for the RequirementFolder [p. 138] indicating a risk value.

NOTE

Use `RMUpdateOpts.getRisk` (getRisk [p. 145]) instead of this method for setting the risk value.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Tests

Gets the Tests [p. 185] collection for the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.Tests`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Methods

CreateCSVTree

`CreateCSVTree` is used to during Import-Export of CSV in Requirement Center. It is used to build the requirements hierarchy. It creates a tree with CSV nodes for the RequirementFolder [p. 138].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
CreateCSVTree(int folderID)
```

CreateRMTree

Creates a tree with `RMRequirement` nodes for the RequirementFolder [p. 138] with the specified RMIntegrationValue [p. 215].

NOTE

This method should only be used if you are performing an integration with a requirements management tool. See Requirements Management Classes [p. 197].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
CreateRMTree(RMIntegrationValue rmv)
```

GetCSVErrorLog

Gets the CSV Error Log associated with the RequirementFolder [p. 138].

Type: `System.Collections.Generic.List<Compuware.QM.QADirector.SDK.CSVError>`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
GetCSVErrorLog()
```

InsertNewRequirement

Inserts a new RequirementNode [p. 146] into the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.RequirementNode`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
InsertNewRequirement()
```

ResetAndDeleteRMIntegration

This method resets the integration with the associated Requirement Management application and deletes all imported requirements from the Requirement Center in *QA*Director for the specified RMIntegrationValue [p. 215].

NOTE

This method should only be used if you are performing an integration with a requirements management tool. See Requirements Management Classes [p. 197].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
ResetAndDeleteRMIntegration(RMIntegrationValue rmv)
```

ResetRMIntegration

This method resets the integration with the associated Requirements Management application for the specified RMIntegrationValue [p. 215].

NOTE

This method should only be used if you are performing an integration with a requirements management tool. See Requirements Management Classes [p. 197].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
ResetRMIntegration(RMIntegrationValue rmv)
```

RMNewEP

This method creates and returns an execution plan with the specified name for the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.ExecutionPlan`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
RMNewEP(string EPName, bool bCreateGroups)
```

- `string EPName` is the name of the **`ExecutionPlan.`**
- `bool bCreateGroups` is used to create execution groups based on your requirement hierarchy. If false, the *Execution Plan* will contain a flat list of tests.

RMReplaceEP

Updates an existing **`ExecutionPlan`** by removing all the nodes in it and recreating the hierarchy from the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.ExecutionPlan`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
RMReplaceEP(int EPDefnID, string EPName, bool bCreateGroups)
```

Parameters

- `int EPDefnID` is the ID of the **`ExecutionPlan`**.
- `string EPName` is the name of the **`ExecutionPlan.`**
- `bool bCreateGroups` is used to create execution groups based on your requirement hierarchy. If false, the *Execution Plan* will contain a flat list of tests.

RMUpdateEP

Updates an existing **`ExecutionPlan`** by removing all the nodes in it and recreating the hierarchy from the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.ExecutionPlan`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
RMUpdateEP(int EPDefnID, string EPName, bool bCreateGroups)
```

Parameters

- `Int EPDefnID` is the ID of the Execution plan.
- `String EPName` is the Name of the Execution plan.
- `bool bCreateGroups` is used to create execution groups based on your requirement hierarchy. If false, the *Execution Plan* will contain a flat list of tests.

UpdateRMTree

Updates all the requirements in a RequirementFolder [p. 138]. Accepts an RMIntegrationValue [p. 215] object and `RMUpdateOpts` object.

NOTE

This method should only be used if you are performing an integration with a requirements management tool. See Requirements Management Classes [p. 197].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
UpdateRMTree(RMIntegrationValue rmv,RMUpdateOpts upOpts)
```

Parameters

- `RMIntegrationValue rmv` - an RMIntegrationValue [p. 215] object.
- `RMUpdateOpts upOpts` - an RMUpdateOpts [p. 144] object.

Classes

RMUpdateOpts

Returns an `RMUpdateOpts` object used to specifiy update options for the RequirementFolder [p. 138].

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Properties

- getResult [p. 144]
- resultFolderID [p. 145]
- getRisk [p. 145]

Properties
getResult

Indicates wheter or not to retrieve results for RMUpdateOpts [p. 144]. Values are:

- `0` - Do not retrieve results.
- `1` - Retrieve results.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder.RMUpdateOpts`

getRisk

Gets/sets whether or not to retrieve risk for RMUpdateOpts [p. 144].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder.RMUpdateOpts`

resultFolderID

Not used for *QA*Director 6.0 and greater.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder.RMUpdateOpts`

RequirementFolders

Returns a collection of RequirementFolder [p. 138] objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 145]
- RequirementFolders [p. 145]

Methods

- Delete [p. 146]
- Exists [p. 146]
- New [p. 146]
- Refresh [p. 196]

Properties

Count

Gets the number of RequirementFolder [p. 138] objects in the RequirementFolders [p. 145] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolders`

RequirementFolders

`RequirementFolders` indexer that returns a RequirementFolder [p. 138] object.

Type: `Compuware.QM.QADirector.SDK.RequirementFolder`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolders`

Syntax

```
RequirementFolders[int ID,bool flag]
```

```
RequirementFolders[int ID]
RequirementFolders[string FolderName]
```

Methods

Delete

Deletes the specified RequirementFolder [p. 138] object in the RequirementFolders [p. 145] collection with the specified `FolderID`.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolders`

Syntax

```
Delete(int FolderID)
```

Exists

Returns a bool indicating whether or not the RequirementFolder [p. 138] object with the supplied `FolderName` exists in the RequirementFolders [p. 145] collection.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolders`

Syntax

```
Exists(string FolderName)
```

New

Creates a new RequirementFolder [p. 138] object in the RequirementFolders [p. 145] collection.

Type: `Compuware.QM.QADirector.SDK.RequirementFolder`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolders`

Syntax

```
New()
New(string FolderName, string Description, bool IsPublic)
```

Refresh

This method refreshes the item(s) from the database.

Type: `void`

Syntax

```
Refresh()
```

RequirementNode

Returns a `RequirementNode` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

Properties

Asset

Gets the TMAsset [p. 187] object which may be a Requirement [p. 132], Test [p. 174] , or Script [p. 164].

Type: `Compuware.QM.QADirector.SDK.TMAsset`

Namespace: `Compuware.QM.QADirector.SDK.RequirementNode`

AssetID

Gets the `AssetID` for the `RequirementNode`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.RequirementNode`

AssetType

Returns the type of custom attrbute.

Type: `Compuware.QACenter.TM.eAssetTypes`. See eAssetTypes [p. 88].

Namespace: `Compuware.QM.QADirector.SDK.RequirementNode`

ChildNodes

Gets the RequirementNodes [p. 148] collection for all child nodes of the current `RequirementNode`.

Type: `Compuware.QM.QADirector.SDK.RequirementNodes`

Namespace: `Compuware.QM.QADirector.SDK.RequirementNode`

CreatedByUser

Returns the User [p. 192] that created the RequirementNode [p. 146].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.RequirmentNode`

ModifiedByUser

Returns the User [p. 192] that last modified the RequirementNode [p. 146].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.RequirmentNode`

Name

Gets the `Name` for the `RequirementNode`.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.RequirementNode`

ParentNode

Gets the parent RequirementNode [p. 146] for the current `RequirementNode`.

Type: `Compuware.QM.QADirector.SDK.RequirementNode`

Namespace: `Compuware.QM.QADirector.SDK.RequirementNode`

ReqRelationID

Gets the requirement relation ID field for the `RequirementNode`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.RequirementNode`

RequirementNodes

Returns a collection of `RequirementNode` objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 148]
- RequirementNodes [p. 148]

Properties

Count

Gets the number of associated requirements in the `RequirementNodes` collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.RequirementNodes`

RequirementNodes

`RequirementNodes` indexer that returns a RequirementNode [p. 146] object.

Type: `Compuware.QM.QADirector.SDK.RequirementNode`

Namespace: `Compuware.QM.QADirector.SDK.RequirementNodes`

Syntax

```
RequirementNodes[string AssetName]
```

Requirements

Returns a collection of Requirement [p. 132] objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

Requirements [p. 149]

Properties

Requirements

`Requirements` indexer that returns a Requirement [p. 132] object.

Type: `Compuware.QM.QADirector.SDK.Requirement`

Namespace: `Compuware.QM.QADirector.SDK.Requirements`

Syntax

```
Requirements[String name]
Requirements[int AssetDefnID]
```

Result

Returns a `Result` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- AssignedTo [p. 150]
- CreatedByUser [p. 150]
- Cycle [p. 150]
- DateCreated [p. 151]
- DateModified [p. 151]
- Description [p. 151]
- EndDate [p. 151]
- EPID [p. 151]
- EPName [p. 151]
- Failed [p. 151]
- JobFolder [p. 152]

- JobID [p. 152]
- JobOwner [p. 152]
- Machine [p. 151]
- Message [p. 152]
- ModifiedByUser [p. 152]
- Name [p. 152]
- NotExecuted [p. 152]
- Passed [p. 152]
- ResultDetailsNode [p. 153]
- ScheduledTime [p. 153]
- ScheduleID [p. 153]
- StartDate [p. 153]
- Status [p. 153]
- StatusEnum [p. 153]
- Total [p. 153]
- Type [p. 153]
- Unexpected [p. 154]
- UnexpectedFailed [p. 154]
- UnexpectedNotExecuted [p. 154]
- UnexpectedPassed [p. 154]

Properties

AssignedTo

Returns a string of either the assigned `User's` username or "`Unassigned`" for the Result [p. 159].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Result`

CreatedByUser

Returns the User [p. 192] that created the Result [p. 159].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.CreatedByUser`

Cycle

Returns the Execution `Cycle` number for the Result [p. 159].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Result`

DateCreated

Returns the date that the Result [p. 159] was created.

Type: `System.DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Result`

DateModified

Returns the date that the Result [p. 159] was last modified.

Type: `System.DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Result`

Description

Returns the description field for the Result [p. 159].

Type: `System.String`

Namespace: `Compuware.QM.QADirector.SDK.Result`

EndDate

Returns the end date for the Result [p. 159].

Type: `System.DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Result`

EPID

Returns the id of the Result [p. 159]'s ExecutionPlan [p. 91].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Result`

EPName

Returns the name of the Result [p. 159]' s ExecutionPlan [p. 91].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Result`

Failed

Returns number of failed scripts in the Result [p. 159].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Result`

Machine

Returns the name of the machine that ran the Job.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Result`

JobFolder

Returns the name of the Job Folder for the Result [p. 159].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Result`

JobID

Returns the id of the Result [p. 159] .

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Result`

JobOwner

Returns the name of User that owns theResult [p. 159]'s Job.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Result`

Message

Returns the Result [p. 159] Message field.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Result`

ModifiedByUser

Returns the User [p. 192] that last modified the Result [p. 159].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Result`

Name

Returns the name of the Result [p. 159].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Result`

NotExecuted

Returns the number of Scripts that were not executed in the Result [p. 159].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Result`

Passed

Returns the number of scripts that passed in the Result [p. 159].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Result`

ResultDetailsNode

Gets the Result Details Root Node (ResultNode [p. 157]) that belong to this Result [p. 159].

Type: `Compuware.QM.QADirector.SDK.ResultNode`

Namespace: `Compuware.QM.QADirector.SDK.Result`

ScheduledTime

Returns the date that the `Job` was scheduled.

Type: `System.DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Result`

ScheduleID

Returns the order id that the Job was scheduled in run.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Result`

StartDate

Returns the date on wihch the Job started.

Type: `System.DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Result`

Status

Returns the status of the Result [p. 159] in a string.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Result`

StatusEnum

Returns the Result [p. 159] Status.

Type: `Compuware.QACenter.TM.eJobStatus`.

Namespace: `Compuware.QM.QADirector.SDK.Result`

Total

Returns the total number of scripts in the Result [p. 159].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Result`

Type

Returns the type of `Job` in a string, either `Automated` or `Manual`.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Result`

Unexpected

Returns the total number of `Unexpected` Results in the Job.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Result`

UnexpectedFailed

Returns the total number of Results that failed unexpectedly.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Result`

UnexpectedNotExecuted

Returns the total number of jobs that did not execute be were supposed to.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Result`

UnexpectedPassed

Returns the total number of Results that passed but were not expected to.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Result`

ResultFolder

Returns a `ResultFolder` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- CreatedByUser [p. 154]
- Description [p. 155]
- ID [p. 155]
- IsPublic [p. 155]
- ModifiedByUser [p. 155]
- Name [p. 155]
- Results [p. 155]

Properties

CreatedByUser

Returns the User [p. 192] that created the ResultFolder [p. 154].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.ResultFolder`

Description

Returns the description of the ResultFolder [p. 154].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ResultFolder`

ID

Returns the id of the ResultFolder [p. 154].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ResultFolder`

IsPublic

Returns a bool as to whether or not the ResultFolder [p. 154] is public.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.ResultFolder`

ModifiedByUser

Returns the User [p. 192] that last modified the ResultFolder [p. 154].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.ResultFolder`

Name

Returns the name of the ResultFolder [p. 154].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ResultFolder`

Results

Returns the collection of Results [p. 160] for the ResultFolder [p. 154].

Type: `Compuware.QM.QADirector.SDK.Results`

Namespace: `Compuware.QM.QADirector.SDK.ResultFolder`

ResultFolders

Returns a collection of ResultFolder [p. 154] objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 156]
- ResultFolders [p. 156]

Methods

- Delete [p. 156]

- Exists [p. 156]
- New [p. 157]
- Refresh [p. 196]

Properties

Count

Gets the number of ResultFolder [p. 154] objects in the ResultFolders [p. 155] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ResultFolders`

ResultFolders

`ResultFolders` indexer that returns a ResultFolder [p. 154] object.

Type: `Compuware.QM.QADirector.SDK.ResultFolder`

Namespace: `Compuware.QM.QADirector.SDK.ResultFolders`

Syntax

```
ResultFolders[string foldername]
```

```
ResultFolders[int ID]
```

Methods

Delete

Deletes the specified ResultFolder [p. 154] object from the ResultFolders [p. 155] collection with the specified `FolderID`.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.ResultFolders`

Syntax

```
Delete(int FolderID)
```

Exists

Returns a bool indicating whether or not the ResultFolder [p. 154] object with the supplied `FolderName` exists in the ResultFolders [p. 155] collection.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.ResultFolders`

Syntax

```
Exists(string FolderName)
```

New

Creates a new ResultFolder [p. 154] object in the ResultFolders [p. 155] collection with the specified `FolderName` and availability.

Type: `Compuware.QM.QADirector.SDK.ResultFolder`

Namespace: `Compuware.QM.QADirector.SDK.ResultFolders`

Syntax

```
New(string FolderName, bool IsPublic)
```

Refresh

This method refreshes the item(s) from the database.

Type: `void`

Syntax

```
Refresh()
```

ResultNode

Returns a `ResultNode` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Asset [p. 157]
- AssetType [p. 158]
- ChildNodes [p. 158]
- CreatedByUser [p. 158]
- EORelationID [p. 158]
- EOScriptRelationID [p. 158]
- ModifiedByUser [p. 158]
- Name [p. 158]
- ParentNode [p. 159]
- Result [p. 159]
- SiblingOrder [p. 160]
- TreeLevel [p. 160]

Properties

Asset

Gets the TMAsset [p. 187] object which may be an ExecutionPlan [p. 91], Group [p. 99] , Test [p. 174] , or Script [p. 164].

Type: `Compuware.QM.QADirector.SDK.TMAsset`

Namespace: `Compuware.QM.QADirector.SDK.ResultNode`

AssetType

Returns the type of ResultNode [p. 157].

Type: `Compuware.QACenter.TM.eAssetTypes`. See eAssetTypes [p. 88].

Namespace: `Compuware.QM.QADirector.SDK.ResultNode`

ChildNodes

Returns the `ResultNodes` collection that are the children of the current ResultNode [p. 157].

Type: `Compuware.QM.QADirector.SDK.ResultNodes`

Namespace: `Compuware.QM.QADirector.SDK.ResultNode`

CreatedByUser

Returns the User [p. 192] that created the ResultNode [p. 157].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.ResultNode`

EORelationID

Returns the `Execution Order Relation ID`. In a result details or Execution Plan tree, each row will have a `EORelationID`. This uniquely identifies a ResultNode [p. 157] within a result.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ResultNode`

EOScriptRelationID

Returns the `Execution Order Script Relation ID`. In a result details or Execution Plan tree, all script nodes will have an `EOScriptRelationID`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ResultNode`

ModifiedByUser

Returns the User [p. 192] that last modified the ResultNode [p. 157].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.ResultNode`

Name

Returns the name of the ResultNode [p. 157].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ResultNode`

ParentNode

Returns the parent ResultNode of the current ResultNode [p. 157].

Type: `Compuware.QM.QADirector.SDK.ResultNode`

Namespace: `Compuware.QM.QADirector.SDK.ResultNode`

Result

Returns a `Result` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- AssignedTo [p. 150]
- CreatedByUser [p. 150]
- Cycle [p. 150]
- DateCreated [p. 151]
- DateModified [p. 151]
- Description [p. 151]
- EndDate [p. 151]
- EPID [p. 151]
- EPName [p. 151]
- Failed [p. 151]
- JobFolder [p. 152]
- JobID [p. 152]
- JobOwner [p. 152]
- Machine [p. 151]
- Message [p. 152]
- ModifiedByUser [p. 152]
- Name [p. 152]
- NotExecuted [p. 152]
- Passed [p. 152]
- ResultDetailsNode [p. 153]
- ScheduledTime [p. 153]
- ScheduleID [p. 153]
- StartDate [p. 153]
- Status [p. 153]
- StatusEnum [p. 153]
- Total [p. 153]

- Type [p. 153]
- Unexpected [p. 154]
- UnexpectedFailed [p. 154]
- UnexpectedNotExecuted [p. 154]
- UnexpectedPassed [p. 154]

SiblingOrder

Returns the sibling order (relative order of the sibling `ResultNode` objects) of the `ResultNode`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ResultNode`

TreeLevel

Returns the number of levels of the tree in the ResultNode [p. 157] .

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ResultNode`

Results

Returns a collection of Result [p. 159] objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 160]
- Results [p. 160]

Methods

Refresh [p. 196]

Properties

Count

Gets the number of Result [p. 159] objects in the Results [p. 160] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Results`

Results

Indexer for `Results` collection to return the specified `Result` object.

Type: `Compuware.QM.QADirector.SDK.Result`

Namespace: `Compuware.QM.QADirector.SDK.Results`

Syntax

```
Results[int ResultID]
```

Methods

Refresh

This method refreshes the item(s) from the database.

Type: `void`

Syntax

```
Refresh()
```

RiskModels

Returns a collection of ProjectRiskModel [p. 128] objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 161]
- RiskModels [p. 161]

Methods

- Clear [p. 162]

Properties

Count

Gets the number of ProjectRiskModel [p. 128] objects in the RiskModels [p. 161] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.RiskModels`

RiskModels

`RiskModels` indexer that returns a `ProjectRiskModel` object.

Type: `Compuware.QM.QADirector.SDK.ProjectRiskModel`

Namespace: `Compuware.QM.QADirector.SDK.RiskModels`

Syntax

```
RiskModels[string name]
RiskModels[int RiskID]
```

Methods

Clear

Clears all ProjectRiskModel [p. 128] objects from the RiskModels [p. 161] collection.

Type: `void`

Namespace: `Compuware.QM.QADirector.SDK.RiskModels`

Syntax

```
Clear()
```

Role

Returns a `Role` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- CreatedByUser [p. 162]
- ID [p. 162]
- ModifiedByUser [p. 162]
- Name [p. 162]

Properties

CreatedByUser

Returns the User [p. 192] that created the Role [p. 162].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Role`

ID

Gets the `ID` value for the Role [p. 162].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Role`

ModifiedByUser

Returns the User [p. 192] that last modified the Role [p. 162].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Role`

Name

Gets the `Name` string for the Role [p. 162].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Role`

Roles

Returns a collection of Role [p. 162] objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 163]
- Roles [p. 163]

Methods

- Exists [p. 163]
- Refresh [p. 196]

Properties

Count

Gets the number of Role [p. 162] objects from the Roles [p. 163] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Roles`

Roles

Indexer for `Roles` collection to return the specified `Role` object.

Type: `Compuware.QM.QADirector.SDK.Role`

Namespace: `Compuware.QM.QADirector.SDK.Roles`

Syntax

```
Roles[int RoleID]
Roles[String name]
```

Methods

Exists

Returns a `bool` as to whether or not a Role [p. 162] object with the specified name exists in the Roles [p. 163] collection.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Roles`

Syntax

```
Exists(string RoleName)
```

Refresh

This method refreshes the item(s) from the database.

Type: `void`

Syntax

```
Refresh()
```

Script

Returns a `Script` object. Inherits from TMAsset [p. 187].

Base class for ManualScript [p. 112].

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- AssetDefnID [p. 165]
- AssetType [p. 189]
- AssociatedDefects [p. 165]
- CreatedByUser [p. 165]
- CustomAttributeValues [p. 165]
- DateCreated [p. 165]
- DateModified [p. 165]
- DefectCount [p. 166]
- Description [p. 166]
- DisplayID [p. 189]
- EstimatedTime [p. 166]
- ID [p. 166]
- IsDirty [p. 166]
- ModifiedByUser [p. 166]
- Name [p. 166]
- ScriptFolder [p. 167]
- ScriptFolderName [p. 167]
- TestingTool [p. 167]
- ToolTypeName [p. 167]

Methods

- AssociateDefect [p. 167]
- ClearFlag [p. 168]
- Update [p. 168]

- UpdateCAs [p. 168]
- UpdateDescription [p. 168]

Properties

AssetDefnID

Returns the assetdefnId field of the Script [p. 164].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Script`

AssetType

Returns the type of asset for the TMAsset [p. 187].

Type: `Compuware.QACenter.TM.eAssetTypes`. See eAssetTypes [p. 88].

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

AssociatedDefects

Returns the Defects [p. 81] collection associated with the Script [p. 164].

Type: `Compuware.QM.QADirector.SDK.Defects`

Namespace: `Compuware.QM.QADirector.SDK.Script`

CreatedByUser

Gets the User [p. 192] who created the Script [p. 164].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Script`

CustomAttributeValues

Returns the CustomAttributeValues [p. 74] collection associated with the Script [p. 164].

Type: `Compuware.QM.QADirector.SDK.CustomAttributeValues`

Namespace: `Compuware.QM.QADirector.SDK.Script`

DateCreated

Returns the date that the Script [p. 164] was created.

Type: `DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Script`

DateModified

Returns the date that the Script [p. 164] was modified.

Type: `DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Script`

DefectCount

Returns the number of defects associated to the Script [p. 164].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Script`

Description

Gets/sets the description of the Script [p. 164].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Script`

DisplayID

Gets the `DisplayID` field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

EstimatedTime

Returns decimal value indicating the estimated time to complete the Script [p. 164].

Type: `decimal`

Namespace: `Compuware.QM.QADirector.SDK.Script`

ID

Gets the ID of the script.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Script`

IsDirty

`Bool` indicating whether or not the Script [p. 164] has changed since initial creation.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Script`

ModifiedByUser

Gets the User [p. 192]who modified the Script [p. 164].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Script`

Name

This property gets/sets the name of the Script [p. 164].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Script`

ScriptFolder

Gets the ScriptFolder [p. 169] that the Script [p. 164] belongs to.

Type: `Compuware.QM.QADirector.SDK.ScriptFolder`

Namespace: `Compuware.QM.QADirector.SDK.Script`

ScriptFolderName

Gets the name of the `ScriptFolder` that the Script [p. 164] belongs to.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Script`

TestingTool

Gets the Tool [p. 189] that the Script [p. 164] uses.

Type: `Compuware.QM.QADirector.SDK.Tool`

Namespace: `Compuware.QM.QADirector.SDK.Script`

ToolTypeName

Gets the name of the Tool [p. 189] used by the Script [p. 164]. Possible values:

- `Automated`
- `Defect`
- `Manual`
- `User Defined`

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Script`

Methods

AssociateDefect

Adds a Defect [p. 79] association to the current Script [p. 164] in a ResultNode [p. 157]. Note that this method can be used only when the Script [p. 164] is in the context of a ResultNode [p. 157]. Also note that this method does not submit the defect to the defect tool. it just sets the defect association between the Script [p. 164] and an existing Defect [p. 79] in the integrated defect tool .

Type: `Compuware.QM.QADirector.SDK.Defect`

Namespace: `Compuware.QM.QADirector.SDK.Script`

Syntax

`AssociateDefect(string DefectToolUniqueId, string DefectToolDisplayId, string DefectSummary)`

- `DefectToolUniqueId` - Defect ID used to uniquely recognize a defect in the defect tool. This could be an internal identifier of the defect tool.

- `DefectToolDisplayId` - Defect id of the defect displayed in the defect tool. For some tools, `DefectToolUniqueId` and `DefectToolDisplayId` could be the same.
- `DefectSummary` - Defect name, title, or summary.

ClearFlag

Clears the dirty flag set on a Script [p. 164]. The "Dirty" flag is set to false when a script is created. Any subsequent changes to the script change the "Dirty" flag to "true". You can read this flag to update a script.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Script`

Syntax

```
ClearFlag()
```

Update

This method updates a Script [p. 164].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Script`

Syntax

- `Update()`
- `Update(bool bIsScriptDirty)`

UpdateCAs

This method updates the CustomAttributeValues [p. 74]collection for the Script [p. 164].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Script`

Syntax

- `UpdateCAs()`

UpdateDescription

This method updates the description field for the Script [p. 164].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Script`

Syntax

- `UpdateDescription()`
- `UpdateDescription(bool bIsScriptDirty)`

ScriptFolder

Returns a `ScriptFolder` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

Methods

Properties

CreatedByUser

Returns the User [p. 192] that created the ScriptFolder [p. 169].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.ScriptFolder`

Description

Gets the `Description` string for the ScriptFolder [p. 169].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ScriptFolder`

ID

Gets the `ID` value for the ScriptFolder [p. 169].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ScriptFolder`

ModifiedByUser

Returns the User [p. 192] that last modified the ScriptFolder [p. 169].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.ScriptFolder`

Name

Gets the `Name` string for the ScriptFolder [p. 169].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.ScriptFolder`

Scripts

Gets the Scripts [p. 172] collection for the ScriptFolder [p. 169].

Type: `Compuware.QM.QADirector.SDK.Scripts`

Namespace: `Compuware.QM.QADirector.SDK.ScriptFolder`

ToolTypeID

Gets the `ToolTypeID` value for the ScriptFolder [p. 169].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ScriptFolder`

Methods

Rename

Renames the current ScriptFolder [p. 169] with the specified name.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.ScriptFolder`

Syntax

```
Rename(string NewFolderName)
```

ScriptFolders

Returns a collection of `ScriptFolder` objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 171]
- ScriptFolders [p. 171]

Methods

- Exists [p. 171]
- New [p. 171]
- Refresh [p. 196]

Properties

Count

Gets the number of ScriptFolder [p. 169] objects in the ScriptFolders [p. 170] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.ScriptFolders`

ScriptFolders

`ScriptFolders` indexer that returns a ScriptFolder [p. 169] object.

Type: `Compuware.QM.QADirector.SDK.ScriptFolder`

Namespace: `Compuware.QM.QADirector.SDK.ScriptFolders`

Syntax

```
ScriptFolders[string foldername]
```

```
ScriptFolders[int FolderID]
```

Methods

Exists

Returns a bool indicating whether or not the ScriptFolder [p. 169] object with the supplied `FolderName` exists in the ScriptFolders [p. 170] collection.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.ScriptFolders`

Syntax

```
Exists(string FolderName)
```

New

Adds a new ScriptFolder [p. 169] object to the ScriptFolders [p. 170] collection with the specified `FolderName` and `Description`.

Type: `Compuware.QM.QADirector.SDK.ScriptFolder`

Namespace: `Compuware.QM.QADirector.SDK.ScriptFolders`

Syntax

```
New(string FolderName, string Description)
```

Refresh

This method refreshes the item(s) from the database.

Type: `void`

Syntax

```
Refresh()
```

Scripts

Provides access to all the Script [p. 164] objects in a given Project [p. 122] in *QA*Director.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 172]
- Script [p. 173]

Methods

- Delete [p. 172]
- Exists [p. 172]
- GetScript [p. 173]
- New [p. 173]
- Refresh [p. 173]

Count

Returns the number of Script [p. 164] objects in the Scripts [p. 172] collection for a project.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Scripts`

Delete

This method deletes a Script [p. 164] from the Scripts [p. 172] collection with the specified id.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Scripts`

Syntax

```
Delete(int scriptdefnid)
```

Parameters

`int scriptdefnid` is the ID of the Script to delete.

Exists

This method checks for the existence of a Script [p. 164] object in the Scripts [p. 172] collection by either the script name or script id, depending on which version of the method used.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Scripts`

Syntax

- **`Exists(string ScriptName)`**
- **`Exists(int Scriptdefnid)`**

GetScript

This method returns a Script [p. 164] object from the Scripts [p. 172] collection with the specified script id.

Type: `Compuware.QM.QADirector.SDK.Script`

Namespace: `Compuware.QM.QADirector.SDK.Scripts`

Syntax

`GetScript(int ScriptDefnID)`

Parameters

`int ScriptDefnID` is the ID of the **`Script.`**

New

This method creates a new Script [p. 164] object in the Scripts [p. 172] collection.

Type: `Compuware.QM.QADirector.SDK.Script`

Namespace: `Compuware.QM.QADirector.SDK.Scripts`

Syntax

- **`New()`**
- **`New(string name, string description)`**

Refresh

This method refreshes the **`Scripts`** collection from the database.

Type: `void`

Namespace: `Compuware.QM.QADirector.SDK.Scripts`

Syntax

`Refresh()`

Return Value

Void

Script

Returns a Script [p. 164] object from the Scripts [p. 172] collection.

Syntax

- **`Script[string scriptname]`**
- **`Script[int scriptdefnid]`**

Returns

`Compuware.QM.QADirector.SDK.Script`

Test

Returns a `Test` object.

Inherits from TMExecAsset [p. 188].

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- AssetDefnID [p. 188]
- AssetType [p. 189]
- AssignedTo [p. 175]
- AssignedToUser [p. 176]
- AssociatedDefects [p. 176]
- AssociatedScripts [p. 176]
- CreatedByUser [p. 176]
- CustomAttributeValuesNonRel [p. 176]
- CustomAttributeValuesRel [p. 176]
- CycleLookUp [p. 176]
- Cycles [p. 176]
- DateCreated [p. 177]
- DefectCount [p. 177]
- Description [p. 177]
- DisplayID [p. 189]
- EstimatedTime [p. 177]
- ExternalRMID [p. 177]
- FailedCount [p. 177]
- InProgressCount [p. 177]
- LastResult [p. 178]
- ModifiedByUser [p. 178]
- Name [p. 178]
- NotExecutableCount [p. 178]

- NotStartedCount [p. 178]
- NotSubmittedCount [p. 178]
- Passed [p. 178]
- ReqRelID [p. 178]
- RequirementCount [p. 179]
- Risk [p. 179]
- RiskLabels [p. 179]
- ScriptCount [p. 179]
- ScriptNodes [p. 179]
- Status [p. 179]
- testdefnid [p. 179]
- TestFolderID [p. 179]

Methods

- AddScript [p. 180]
- AssociateDefect [p. 180]
- RemoveScript [p. 180]
- SaveTestProperties [p. 181]
- Update [p. 181]
- UpdateLight [p. 181]

Properties

AssetDefnID

Gets the asset ID value of the TMAsset [p. 187].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

AssetType

Returns the type of asset for the TMAsset [p. 187].

Type: `Compuware.QACenter.TM.eAssetTypes`. See eAssetTypes [p. 88].

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

AssignedTo

Gets the name of the user to whom the Test [p. 174] is assigned.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Test`

AssignedToUser

Returns the User [p. 192] assigned to the Test [p. 174].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Test`

AssociatedDefects

Returns the Defects [p. 81] collection assigned to the Test [p. 174].

Type: `Compuware.QM.QADirector.SDK.Defects`

Namespace: `Compuware.QM.QADirector.SDK.Test`

AssociatedScripts

Returns the Scripts [p. 172] collection associated with the Test [p. 174].

Type: `Compuware.QM.QADirector.SDK.Defects`

Namespace: `Compuware.QM.QADirector.SDK.Test`

CreatedByUser

Returns the User [p. 192] who created the Test [p. 174].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Test`

CustomAttributeValuesNonRel

Returns the collection of non-relational CustomAttributeValues [p. 74] of the Test [p. 174].

Type: `Compuware.QM.QADirector.SDK.CustomAttributeValues`

Namespace: `Compuware.QM.QADirector.SDK.Test`

CustomAttributeValuesRel

Returns the collection of relational CustomAttributeValues [p. 74] of the Test [p. 174].

Type: `Compuware.QM.QADirector.SDK.CustomAttributeValues`

Namespace: `Compuware.QM.QADirector.SDK.Test`

CycleLookUp

Returns a string array of the Cycles belonging to the current Test [p. 174].

Type: `string[]`

Namespace: `Compuware.QM.QADirector.SDK.Test`

Cycles

Returns the list of cycles for the Test [p. 174].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Test`

DateCreated

Returns the date that the Test [p. 174] was created.

Type: `DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Test`

DefectCount

Gets the count of defects associated with the Test [p. 174].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Test`

Description

Gets the description of the Test [p. 174] in a string.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Test`

DisplayID

Gets the `DisplayID` field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

EstimatedTime

Returns a decimal of the estimated time for the Test [p. 174].

Type: `decimal`

Namespace: `Compuware.QM.QADirector.SDK.Test`

ExternalRMID

Gets the ExternalRIMID string for the Test [p. 174].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Test`

FailedCount

Returns the number of children items with a status of `Failed` for the Test [p. 174].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Test`

InProgressCount

Returns the number of children items with a status of `In Progress` for the Test [p. 174].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Test`

LastResult

Gets the last result of the Test [p. 174] in a string.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Test`

ModifiedByUser

Returns the User [p. 192] who last modified the Test [p. 174].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Test`

Name

Gets the name of the Test [p. 174].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Test`

NotExecutableCount

Returns the number of children items with a status of `Not Executable` for the Test [p. 174].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Test`

NotStartedCount

Returns the number of children items with a status of `Not Started` for the Test [p. 174].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Test`

NotSubmittedCount

Returns the number of children items with a status of `Not Submitted` for the Test [p. 174].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Test`

Passed

Returns the number of children items with a status of `Passed` for the Test [p. 174].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Test`

ReqRelID

Returns the related requirement ID for the Test [p. 174].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Test`

RequirementCount

Gets the count of requirements associated to the Test [p. 174].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Test`

Risk

Gets the risk of a Test [p. 174].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Test`

RiskLabels

Returns a string array of the Risk Labels belonging to the current Test [p. 174].

Type: `string[]`

Namespace: `Compuware.QM.QADirector.SDK.Test`

ScriptCount

This property returns the number of scripts associated to the Test [p. 174].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Test`

ScriptNodes

Gets an ArrayList of scripts associated with the Test [p. 174].

Type: `System.Collections.ArrayList`

Namespace: `Compuware.QM.QADirector.SDK.Test`

Status

Gets/sets the status of the Test [p. 174] with either of the following values:

- `Complete`
- `In Progress`

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Test`

testdefnid

Gets the test definition ID for the Test [p. 174].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Test`

TestFolderID

Gets the `TestFolder`'s id that the Test [p. 174] belongs to.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Test`

Methods

AddScript

This method associates a Script [p. 164] with a Test [p. 174].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Test`

Syntax

```
AddScript(Script srpt)
```

Parameters

`Script srpt` is a `Compuware.QM.QADirector.SDK.Script` object.

AssociateDefect

Adds a Defect [p. 79] association to the current Test [p. 174] in a ResultNode [p. 157]. Note that this method can be used only when the Test [p. 174] is in the context of a ResultNode [p. 157]. Also note that this method does not submit the defect to the defect tool. it just sets the defect association between the `Test` and an existing `Defect` in the integrated defect tool .

Type: `Compuware.QM.QADirector.SDK.Defect`

Namespace: `Compuware.QM.QADirector.SDK.Test`

Syntax

```
AssociateDefect(string DefectToolUniqueId, string DefectToolDisplayId, string DefectSummary)
```

- `DefectToolUniqueId` - Defect ID used to uniquely recognize a defect in the defect tool. This could be an internal identifier of the defect tool.
- `DefectToolDisplayId` - Defect id of the defect displayed in the defect tool. For some tools, `DefectToolUniqueId` and `DefectToolDisplayId` could be the same.
- `DefectSummary` - Defect name, title, or summary.

RemoveScript

This method removes the association between the Test [p. 174] and a script.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Test`

Syntax

```
RemoveScript(Script srpt)
```

Parameters

`Script srpt` is a `Compuware.QM.QADirector.SDK.Script` object.

SaveTestProperties

Saves properties of the Test [p. 174].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Test`

Syntax

```
SaveTestProperties()
```

Update

Updates a Test [p. 174] with all the associated scripts.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Test`

Syntax

```
Update()
```

UpdateLight

This method updates a Test [p. 174] with the new name and description.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Test`

Syntax

```
UpdateLight()
```

TestFolder

Returns a `TestFolder` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- CreatedByUser [p. 182]
- Description [p. 182]
- ExecutionPlans [p. 182]
- ID [p. 182]
- IsPublic [p. 182]
- ModifiedByUser [p. 182]
- Name [p. 182]

- Tests [p. 183]

Methods

CreateEP [p. 183]

Properties

CreatedByUser

Returns the User [p. 192] that created the TestFolder [p. 181].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.TestFolder`

Description

Gets the `Description` string for the TestFolder [p. 181].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TestFolder`

ExecutionPlans

Gets the ExecutionPlans [p. 96] collection for the TestFolder [p. 181].

Type: `Compuware.QM.QADirector.SDK.ExecutionPlans`

Namespace: `Compuware.QM.QADirector.SDK.TestFolder`

ID

Gets the `ID` value for the TestFolder [p. 181].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.TestFolder`

IsPublic

Gets a bool indicating whether or not the TestFolder [p. 181] is public.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.TestFolder`

ModifiedByUser

Returns the User [p. 192] that last modified the TestFolder [p. 181].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.TestFolder`

Name

Gets the `Name` string for the TestFolder [p. 181].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TestFolder`

Tests

Gets the Tests [p. 185] collection associated with the TestFolder [p. 181].

Type: `Compuware.QM.QADirector.SDK.Tests`

Namespace: `Compuware.QM.QADirector.SDK.TestFolder`

Methods

CreateEP

`CreateEP` creates a new ExecutionPlan [p. 91] with the given array of Tests in the TestFolder [p. 181]. This method does not replace or update an existing `ExecutionPlan`.

NOTE

`CreateEP` returns `null` if it fails. Call `GetLastError` to return the error string.

Type: `Compuware.QM.QADirector.SDK.ExecutionPlan`

Namespace: `Compuware.QM.QADirector.SDK.TestFolder`

Syntax

`CreateEP(string EPName, string EPDescription, Test[] TestArray)`

```
//...
int TestID;
Test t = null;
Test[] TestArray = new Test[this.dgViewTests.SelectedRows.Count];
int i=0;
foreach (DataGridViewRow SelRow in this.dgViewTests.SelectedRows)
{
     TestID = Convert.ToInt32(SelRow.Cells[0].Value);
     t = tstfolder.Tests[TestID];
     TestArray[i] = t;
     i++;
}

ExecutionPlan ep = null;
if (TestArray.Length > 0)
{
     ep = tstfolder.CreateEP(EPName, EPDesc, TestArray);
}
 // ...
```

TestFolders

Returns a collection of `TestFolder` objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 184]
- TestFolders [p. 184]

Methods

- Delete [p. 184]
- Exists [p. 184]
- New [p. 185]
- Refresh [p. 196]

Properties

Count

Gets the number of TestFolder [p. 181] objects in the TestFolders [p. 183] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.TestFolders`

TestFolders

`TestFolders` indexer that returns a TestFolder [p. 181] object.

Type: `Compuware.QM.QADirector.SDK.TestFolder`

Namespace: `Compuware.QM.QADirector.SDK.TestFolders`

Syntax

```
TestFolders[string foldername]
```

```
TestFolders[int ID]
```

Methods

Delete

Deletes the specified TestFolder [p. 181] object in the TestFolders [p. 183] collection with the specified `FolderID`.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.TestFolders`

Syntax

```
Delete(int FolderID)
```

Exists

Returns a bool indicating whether or not the TestFolder [p. 181] object with the supplied `FolderName` exists in the TestFolders [p. 183] collection.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.TestFolders`

Syntax

```
Exists(string FolderName)
```

New

Creates a new TestFolder [p. 181] object in the TestFolders [p. 183] collection with the specified `FolderName` and availability.

Type: `Compuware.QM.QADirector.SDK.TestFolder`

Namespace: `Compuware.QM.QADirector.SDK.TestFolders`

Syntax

```
New(string FolderName, bool IsPublic)
```

Refresh

This method refreshes the item(s) from the database.

Type: `void`

Syntax

```
Refresh()
```

Tests

Provides access to all the tests in a given project in *QA*Director.

Namespace: `Compuware.QM.QADirector.SDK`

Methods

- Delete [p. 186]
- Exists [p. 186]
- GetAsset [p. 186]
- New [p. 186]
- Refresh [p. 186]
-

Properties

- Count [p. 185]
- Test [p. 187]

Count

Gets the number of Test [p. 174] objects in the Tests [p. 185] collection for a project.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Tests`

Delete

Deletes one or more Test [p. 174] objects from the Tests [p. 185] collection.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Tests`

Syntax

```
Delete(string testids)
```

string testids

This is a comma separated string of test ids to delete.

Exists

Checks for the existence of a Test [p. 174] object in the Tests [p. 185] collection with the specified `Name`.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Tests`

Syntax

```
Exists(string TestName)
```

GetAsset

Returns a Test [p. 174] object with the specified test id from the Tests [p. 185] collection.

Type: `Compuware.QM.QADirector.SDK.Test`

Namespace: `Compuware.QM.QADirector.SDK.Tests`

Syntax

```
GetAsset(int AssetDefnID)
```

Parameters

`int AssetDefnID` is the asset ID.

New

Creates and returns a new Test [p. 174] object in the Tests [p. 185] collection.

Type: `Compuware.QM.QADirector.SDK.Test`

Namespace: `Compuware.QM.QADirector.SDK.Tests`

Syntax

```
New()
```

Refresh

Refreshes the **`Tests`** collection in the database.

Type: `void`

Namespace: `Compuware.QM.QADirector.SDK.Tests`

Syntax

```
Refresh()
```

Test

Gets a Test [p. 174] object with either the supplied asset name or asset id.

Syntax

- **`Test[int assetdefnid]`** returns a **`Test`** object by `assetdefnid`
- **`Test[string assetname]`** returns a **`Test`** object by test `assetname`.

Type: `Compuware.QM.QADirector.SDK.Test`

Namespace: `Compuware.QM.QADirector.SDK.Tests`

TMAsset

Returns a `TMAsset` object.

`TMAsset` is the base class of TMExecAsset [p. 188], Requirement [p. 132], ExecutionPlan [p. 91], and Script [p. 164].

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- AssetDefnID [p. 188]
- AssetType [p. 189]
- Description [p. 189]
- DisplayID [p. 189]
- Name [p. 189]

Properties

AssetDefnID

Gets the asset ID value of the TMAsset [p. 187].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

AssetType

Returns the type of asset for the TMAsset [p. 187].

Type: `Compuware.QACenter.TM.eAssetTypes`. See eAssetTypes [p. 88].

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

Description

Gets the description field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

DisplayID

Gets the `DisplayID` field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

Name

Gets the `Name` field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

TMExecAsset

Returns a `TMExecAsset` object. Inherits from TMAsset [p. 187]. `TMExexAsset` is the base class for:

1. Test [p. 174]
2. Group [p. 99]

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- AssetDefnID [p. 188]
- AssetType [p. 189]
- CreatedByUser [p. 189]
- Description [p. 189]
- DisplayID [p. 189]
- ModifiedByUser [p. 189]
- Name [p. 189]

Properties

AssetDefnID

Gets the asset ID value of the TMAsset [p. 187].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

AssetType

Returns the type of asset for the TMAsset [p. 187].

Type: `Compuware.QACenter.TM.eAssetTypes`. See eAssetTypes [p. 88].

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

CreatedByUser

Returns the User [p. 192] that created the TMExecAsset [p. 188].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.TMExecAsset`

Description

Gets the description field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

DisplayID

Gets the `DisplayID` field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

ModifiedByUser

Returns the User [p. 192] that last modified the TMExecAsset [p. 188].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.TMExecAsset`

Name

Gets the `Name` field of the TMAsset [p. 187].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.TMAsset`

Tool

Returns a `Tool` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- CreatedByUser [p. 190]
- EditScript [p. 190]
- HelpCommands [p. 190]
- LaunchTool [p. 190]

- ModifiedByUser [p. 190]
- RetrieveScript [p. 191]
- RunScript [p. 191]
- ScriptFolders [p. 191]
- ScriptParameters [p. 191]
- ScriptPassValue [p. 191]
- ToolExtensions [p. 191]
- ToolName [p. 191]
- ToolType [p. 191]
- ToolTypeID [p. 192]
- WebServiceURL [p. 192]

Properties

CreatedByUser

Returns the User [p. 192] that created the Tool [p. 189].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Tool`

EditScript

Returns the `EditScript` string for the Tool [p. 189].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Tool`

HelpCommands

Returns the `HelpCommands` string for the Tool [p. 189].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Tool`

LaunchTool

Returns the `LaunchTool` string for the Tool [p. 189].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Tool`

ModifiedByUser

Returns the User [p. 192] that last modified the Tool [p. 189].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Tool`

RetrieveScript

Returns the `RetrieveScript` string for the Tool [p. 189].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Tool`

RunScript

Returns the `RunScript` string for the Tool [p. 189].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Tool`

ScriptFolders

Returns the ScriptFolders [p. 170] object collection associated with the Tool [p. 189].

Type: `Compuware.QM.QADirector.SDK.ScriptFolders`

Namespace: `Compuware.QM.QADirector.SDK.Tool`

ScriptParameters

Returns the `ScriptParameters` string for the Tool [p. 189].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Tool`

ScriptPassValue

Returns the `ScriptPassValue` string for the Tool [p. 189].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Tool`

ToolExtensions

Returns the `ToolExtensions` string for the Tool [p. 189].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Tool`

ToolName

Returns the `ToolName` string for the Tool [p. 189].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Tool`

ToolType

Returns the `ToolType` string for the Tool [p. 189].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Tool`

ToolTypeID

Returns the `ToolTypeID` for the Tool [p. 189].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Tool`

WebServiceURL

Returns the `WebServiceURL` string for the Tool [p. 189].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Tool`

Tools

Returns a collection of Tool [p. 189] objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 192]
- Tools [p. 192]

Properties

Count

Gets the number of Tool [p. 189] objects in the Tools [p. 192] collection for the current Client [p. 60].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Tools`

Tools

`Tools` indexer that returns a Tool [p. 189] object.

Type: `Compuware.QM.QADirector.SDK.Tool`

Namespace: `Compuware.QM.QADirector.SDK.Tools`

Syntax

```
Tools[String name]
Tools[int ToolTypeID]
```

User

Returns a `User` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

Methods

Properties

CreatedByUser

Returns the User [p. 192] that created the User [p. 192].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.User`

Email

Gets the `Email` string for the User [p. 192].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.User`

FirstName

Gets the `FirstName` string for the User [p. 192].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.User`

IsAdmin

Returns a bool indicating whether or not the User [p. 192] has `Administrator` privileges.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.User`

LastName

Gets the `LastName` string for the User [p. 192].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.User`

ModifiedByUser

Returns the User [p. 192] that last modified the User [p. 192].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.User`

RoleID

Gets the `RoleID` `int` for the User [p. 192].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.User`

UserID

Gets the `UserID` `int` for the User [p. 192].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.User`

UserName

Gets the `UserName` string for the User [p. 192].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.User`

Methods

HasPermissions

Validates user permissions based on the list of available product permissions for the User [p. 192].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.User`

Syntax

`HasPermissions(int PermissionID)`

`int PermissionID` - Use available list of permissions, Example:

```
HasPermissions(RolesPermissions.ManageProjects)
```

HasProjectPermissions

Validates user permissions for a project based on the list of available product permissions for the User [p. 192].

NOTE

This method should be used only when there is a need to check for a project-specific permission without opening the project. For all other needs, use HasPermissions [p. 194] method.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.User`

Syntax

`HasProjectPermissions(int ProjectID, int PermissionID)`

`int ProjectID` - ID of the project in which you want to check the permission.

`int PermissionID` - Use available list of permissions, Example:

```
HasProjectPermissions(iProjectID, RolesPermissions.ManageProjects)
```

Users

Returns a collection of User [p. 192] objects.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 195]
- Users [p. 195]

Methods

- Delete [p. 196]
- Exists [p. 196]
- New [p. 196]

Properties

Count

Gets the total number of User [p. 192] objects in the Users [p. 195] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Users`

Users

`Users` indexer that returns a User [p. 192] object.

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Users`

Syntax

`Users[string UserName]`

```
Users[int UserID]
```

Methods

Delete

Deletes a User [p. 192] with the specified id from the Users [p. 195] collection.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Users`

Syntax

```
Delete(int UserID)
```

Exists

Checks the Users [p. 195] collection to see if a User [p. 192] exists with the specified name.

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.Users`

Syntax

```
Exists(string UserName)
```

New

Creates and returns a new User [p. 192] object with the specified parameters in the Users [p. 195] collection.

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Users`

Syntax

```
New(int RoleId, string UserName, string Password, string FistName, string
LastName, string Department, string Phone, string Email, bool IsAdmin)
```

Common Methods

Refresh

This method refreshes the item(s) from the database.

Type: `void`

Syntax

```
Refresh()
```

Requirements Management Classes

The following classes are used specifically to integrate a requirements management application with *QA*Director. They should not be used unless you are doing this.

- Node [p. 197]
- Nodes [p. 206]
- RMFolder [p. 209]
- RMIntegrationValue [p. 215]
- RMIntegrationValues [p. 217]

Overview of Creating an RM Integration

The following are the steps to setup an RMIntegration with *QA*Director for any RM tool:

1. Create a connection.
2. Get list of clients.
3. Open a client.
4. Get list of projects.
5. Open a project.
6. Get list of Requirement Folders or Create a new Requirement Folder
7. Create an `RMIntegration`:

```
RequirementFolder reqFolder = curProject.RequirementFolders[folderID];

RMIntegrationValue rmv = reqFolder.RMIntegrationValues.New
("Optimal Trace Enterprise", 11, "business Requirements", "", "", "", "", "");
```

8. Build the `Nodes` structure (Refer to the Sample code: **btnRCBuildReqs_Click()**)
9. Create/Update requirements in QADirector using:

```
reqFolder.CreateRMTree(...)

//or

reqFolder.UpdateRMTree(...)
```

10. To disconnect the `RMIntegration` for a `RequirementFolder`, use either:

```
 reqFolder.ResetRMIntegration(...)

//or

reqFolder.ResetAndDeleteRMIntegration(...) //(Refer to the sample code)
```

For more information, see Building an RM Node Collection Example [p. 208].

Node

Returns a `Node` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

Properties

ActualResult

Retuns the one of the following possiblr `ActualResult` values for a Node [p. 197]:

- `Not Started`
- `Passed`
- `Failed`
- `Not Executable`
- `In Progress`
- `Not Submitted`
- `PassedForced`
- `FailedForced`
- `NotExecutableForced`

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Node`

Asset

Retuns the TMAsset [p. 187] object for the Node [p. 197].

Type: `Compuware.QM.QADirector.SDK.TMAsset`

Namespace: `Compuware.QM.QADirector.SDK.Node`

AssetDefintionID

Returns the `Asset Defition ID` of the Node [p. 197]. This is only available after an update to the database.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

AssetType

Returns the type of asset of the Node [p. 197].

Type: `Compuware.QACenter.TM.eAssetTypes`. See eAssetTypes [p. 88].

Namespace: `Compuware.QM.QADirector.SDK.Node`

AssetTypeAsNumber

Reruns the TMAsset [p. 187] number for the Node [p. 197].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

AssociatedDefects

Reruns the Defects [p. 81] collection object for the Node [p. 197].

Type: `Compuware.QM.QADirector.SDK.Defects`

Namespace: `Compuware.QM.QADirector.SDK.Node`

AssociatedScripts

Reruns the AssociatedScripts [p. 58] collection object for the Node [p. 197].

Type: `Compuware.QM.QADirector.SDK.AssociatedScripts`

Namespace: `Compuware.QM.QADirector.SDK.Node`

CoveragePercent

Reruns the coverage percentage value for the Node [p. 197].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

CreatedByUser

Returns the User [p. 192] that created the Node [p. 197].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Node`

CustomAttributeValues

Returns the CustomAttributeValues [p. 74] collection for the Node [p. 197].

Type: `Compuware.QM.QADirector.SDK.CustomAttributeValues`

Namespace: `Compuware.QM.QADirector.SDK.Node`

Cycles

Retuns the Cycles [p. 77] collection object for the Node [p. 197].

Type: `Compuware.QM.QADirector.SDK.Cycles`

Namespace: `Compuware.QM.QADirector.SDK.Node`

DateCreated

Returns the date that the Node [p. 197] was created.

Type: `System.DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Node`

DateModified

Returns the date that the Node [p. 197] was last modified.

Type: `System.DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Node`

DefectCount

Retuns the number of defects associated with the Node [p. 197].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

Description

Returns the description field for the Node [p. 197].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Node`

DisplayID

Returns the display id field for the Node [p. 197].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

ElapsedTime

Gets the elapsed time for a Node [p. 197] of type `Test`.

Type: `System.TimeSpan`

Namespace: `Compuware.QM.QADirector.SDK.Node`

EndTime

Returns the end time information for a Node [p. 197] of type `Test`.

Type: `System.DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Node`

EstimatedTime

Returns the estimated time in hours for a Node [p. 197] of type `Test`.

Type: `double`

Namespace: `Compuware.QM.QADirector.SDK.Node`

ExpectedResult

Returns the expected result value for the Node [p. 197]:

- `Not Started`
- `Passed`
- `Failed`
- `Not Executable`
- `In Progress`
- `Not Submitted`
- `PassedForced`
- `FailedForced`
- `NotExecutableForced`

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Node`

ExternalID

Gets or sets the external tool's requirement ID for the Node [p. 197].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Node`

ExternalParentID

Gets or sets the external parent ID for the Node [p. 197].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Node`

FailedCount

Returns the failed count value (number of failed tests) for the Node [p. 197].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

FailureDescription

Returns the failure message for a Node [p. 197] of type `Test`.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Node`

Folder

Returns the RequirementFolder [p. 138] object related to the the Node [p. 197].

Type: `Compuware.QM.QADirector.SDK.RequirementFolder`

Namespace: `Compuware.QM.QADirector.SDK.Node`

InProgressCount

Returns the number of tests with an `InProgress` status for the Node [p. 197].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

JobMachine

Returns the execution machine information for a Node [p. 197] of type `Type`.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Node`

LastResult

Returns the last result value for the Node [p. 197]:

- `Not Started`
- `Passed`
- `Failed`
- `Not Executable`
- `In Progress`
- `Not Submitted`
- `PassedForced`
- `FailedForced`
- `NotExecutableForced`

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Node`

LastResultUpdate

Returns the date that a Node [p. 197] of type `Test` was last updated.

Type: `System.DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Node`

ModifiedByUser

Returns the User [p. 192] that last modified the Node [p. 197].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.Node`

Name

Returns the name of the Node [p. 197].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Node`

Nodes

Returns the Nodes [p. 206] collection belonging to the Node [p. 197].

Type: `Compuware.QM.QADirector.SDK.Nodes`

Namespace: `Compuware.QM.QADirector.SDK.Node`

NotExecutableCount

Returns the number of `Tests` in the Node [p. 197] that have the status of `NotExecutable`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

NotStartedCount

Returns the number of `Tests` in the Node [p. 197] that have the status of `NotStarted`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

NotSubmittedCount

Returns the number of `Tests` in the Node [p. 197] that have the status of `NotSubmitted`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

PassedCount

Returns the number of `Tests` in the Node [p. 197] that have the status of `Passed`.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

ReqParentRelationID

Returns the Req Parent Relation ID of the Node [p. 197]. This is only available after an update to repository

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

ReqRelationID

Returns the requirement relation ID of the Node [p. 197]. This is only available after an update to repository

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

Risk

Returns the risk value of the Node [p. 197] as a number.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

RiskLabel

Returns the risk label of the Node [p. 197] as a string.

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.Node`

RiskModelAttributeValues

Returns the `RiskModelAttributeValues` (CustomAttributeValues [p. 74]) collection of the Node [p. 197].

Type: `Compuware.QM.QADirector.SDK.CustomAttributeValues`

Namespace: `Compuware.QM.QADirector.SDK.Node`

ScriptCount

Returns the number of scripts in the Node [p. 197] .

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

StartTime

Returns the start time for a Node [p. 197] of type `Test`.

Type: `System.DateTime`

Namespace: `Compuware.QM.QADirector.SDK.Node`

Status

Returns the `Status` of the Node [p. 197].

Type: `Compuware.QACenter.TM.eAssetStatus`. See eAssetStatus [p. 88].

Namespace: `Compuware.QM.QADirector.SDK.Node`

Tag

Gets/sets the tag for the Node [p. 197].

Type: `System.object`

Namespace: `Compuware.QM.QADirector.SDK.Node`

TestCount

Returns the number of tests in the Node [p. 197].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

UserAssignedTo

Returns the number value of the user who is assigned to the Node [p. 197].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Node`

Nodes

The `Nodes` class keeps an ordered collection of Node [p. 197] objects. The hierarchy is maintained in this collection.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- Count [p. 206]
- Nodes [p. 206]

Methods

- Clear [p. 207]
- NewCSVReqNode [p. 207]
- NewCSVTestNode [p. 207]
- NewRMReqNode [p. 207]
- NewRMTestNode [p. 208]

Code Sample

Building an RM Node Collection Example [p. 208]

Properties

Count

Returns the count of all Node [p. 197] objects (including Test and Requirement) from the Nodes [p. 206] collection.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.Nodes`

Nodes

`Nodes` indexer that returns a Node [p. 197] object.

Type: `Compuware.QM.QADirector.SDK.Node`

Namespace: `Compuware.QM.QADirector.SDK.Nodes`

Syntax

```
Nodes[int ID]
Nodes[String IDName]
```

Methods

Clear

Deletes all Node [p. 197] objects from the Nodes [p. 206] collection.

Type: `void`

Namespace: `Compuware.QM.QADirector.SDK.Nodes`

Syntax

```
Clear()
```

NewCSVReqNode

Creates a new `Node` of type *CSVReq*.

Type: `Compuware.QM.QADirector.SDK.Node`

Namespace: `Compuware.QM.QADirector.SDK.Nodes`

Syntax

```
NewCSVReqNode(string name, bool ignoreFlag)
```

NewCSVTestNode

Creates a new `Node` of type *CSVTest*.

Type: `Compuware.QM.QADirector.SDK.Node`

Namespace: `Compuware.QM.QADirector.SDK.Nodes`

Syntax

```
NewCSVTestNode(string name, bool ignoreFlag)
```

NewRMReqNode

Creates a new Requirement `Node` object.

Type: `Compuware.QM.QADirector.SDK.Node`

Namespace: `Compuware.QM.QADirector.SDK.Nodes`

Syntax

```
NewRMReqNode(string ExternalID)
```

Parameters

`string ExternalID` is the External ID of the third party Requirement.

NewRMTestNode

Creates a new `Node` of type *Test*.

Type: `Compuware.QM.QADirector.SDK.Node`

Namespace: `Compuware.QM.QADirector.SDK.Nodes`

Syntax

`NewRMTestNode(string ExternalID)`

Parameters

`string ExternalID` is the External ID of the third party Requirement.

Building an RM Node Collection Example

```
Node root, r1, r2, t1, t2;
//get the id of the selected requirement folder
int folderID = Convert.ToInt32(GetSelectedRowID(dgViewReqFolders));

reqFolder = curProject.RequirementFolders[folderID];
reqFolder.Nodes.Clear();
nodes = reqFolder.Nodes;

//GetNextAlphaID() - gives the unique id of the requirement.
//In the case of RM Tool, this would be the unique requirement ID of the external tool
root = nodes.NewRMReqNode(GetNextAlphaID());
root.Name = "Req 1";
root.Description = "Req 1 descrip";

#region Level 2
r1 = root.Nodes.NewRMReqNode(GetNextAlphaID());
r1.Name = "Req 1.1";
r1.Description = "MY Req 1.1 descrip";
r1.Risk = 4;
#endregion

r2 = root.Nodes.NewRMReqNode(GetNextAlphaID());
r2.Name = "Req 1.2";
r2.Description = "MY Req 1.2 descrip";
r2.Risk = 3;
r2.EstimatedTime = (double)10;

#region Level 4 - Tests
t1 = r1.Nodes.NewRMTestNode(GetNextAlphaID());
t1.Name = "Test1";
t1.Description = "MY Test1 descrip";
t1.EstimatedTime = (double)5;


//code to import data into QADirector

RMIntegrationValue rmv;
if (rmv == null)
{
      rmv = reqFolder.RMIntegrationValues.GetRMIntegrationValue("Optimal Trace
Enterprise",
 11, "business Requirements", "", "", "");
}

bool blnVal = reqFolder.CreateRMTree(rmv);

//code to update data in QADirector
RMUpdateOpts ropts = new RMUpdateOpts();
ropts.getResult = (int)eRMResultChoice.All;
ropts.getRisk = true;
RMIntegrationValue rmv;
if (rmv == null)
{
```

```
      rmv = reqFolder.RMIntegrationValues.GetRMIntegrationValue("Optimal Trace
Enterprise",
 11, "business Requirements", "", "", "");
}
bool blnVal = reqFolder.UpdateRMTree(rmv, ropts);
```

RMFolder

`RMFolder` is a key class for requirements management integration. It represents the `Default Requirement Folder` in the *QA*Director application. All of the data push/pull is done through this class.

`RMFolder` inherits from RequirementFolder [p. 138].

Namespace: `Compuware.QM.QADirector.SDK`

Properties

- CreatedByUser [p. 210]
- Description [p. 210]
- ExecutionPlans [p. 210]
- ID [p. 210]
- IsPublic [p. 210]
- ModifiedByUser [p. 210]
- Name [p. 210]
- Nodes [p. 210]
- RequirementNodes [p. 211]
- Requirements [p. 211]
- RMIntegrationValues [p. 211]
- SetRMUpdateOptsGetRiskValue [p. 212]
- Tests [p. 212]

Methods

- CreateCSVTree [p. 212]
- CreateRMTree [p. 212]
- GetCSVErrorLog [p. 212]
- InsertNewRequirement [p. 213]
- ResetAndDeleteRMIntegration [p. 213]
- ResetRMIntegration [p. 213]
- RMNewEP [p. 214]
- RMReplaceEP [p. 214]
- RMUpdateEP [p. 211]
- UpdateRMTree [p. 214]

Properties

CreatedByUser

Returns the User [p. 192] that created the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Description

Gets the description field for the RequirementFolder [p. 138].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

ExecutionPlans

Gets the ExecutionPlans [p. 96] associated with the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.ExecutionPlans`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

ID

Gets the id value for the RequirementFolder [p. 138].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

IsPublic

Returns a `bool` indicating the availability of the RequirementFolder [p. 138].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

ModifiedByUser

Returns the User [p. 192] that last modified the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Name

Gets the name for the RequirementFolder [p. 138].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Nodes

Gets the Nodes [p. 206] collection for the RequirementFolder [p. 138].

NOTE

This property should only be used if you are performing an integration with a requirements management tool. See Requirements Management Classes [p. 197].

Type: `Compuware.QM.QADirector.SDK.Nodes`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

RequirementNodes

Gets the RequirementNodes [p. 148] collection for the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.RequirementNodes`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Requirements

Gets the Requirements [p. 149] collection for the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.Requirements`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

RMIntegrationValues

Gets the RMIntegrationValues [p. 217] collection for the RequirementFolder [p. 138].

NOTE

This property should only be used if you are performing an integration with a requirements management tool. See Requirements Management Classes [p. 197].

Type: `Compuware.QM.QADirector.SDK.Nodes`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

RMUpdateEP

Updates an existing **`ExecutionPlan`** by removing all the nodes in it and recreating the hierarchy from the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.ExecutionPlan`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
RMUpdateEP(int EPDefnID, string EPName, bool bCreateGroups)
```

Parameters

- `Int EPDefnID` is the ID of the Execution plan.
- `String EPName` is the Name of the Execution plan.
- `bool bCreateGroups` is used to create execution groups based on your requirement hierarchy. If false, the *Execution Plan* will contain a flat list of tests.

SetRMUpdateOptsGetRiskValue

Sets a value for the RequirementFolder [p. 138] indicating a risk value.

NOTE

Use `RMUpdateOpts.getRisk` (getRisk [p. 145]) instead of this method for setting the risk value.

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Tests

Gets the Tests [p. 185] collection for the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.Tests`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Methods

CreateCSVTree

`CreateCSVTree` is used to during Import-Export of CSV in Requirement Center. It is used to build the requirements hierarchy. It creates a tree with CSV nodes for the RequirementFolder [p. 138].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
CreateCSVTree(int folderID)
```

CreateRMTree

Creates a tree with `RMRequirement` nodes for the RequirementFolder [p. 138] with the specified RMIntegrationValue [p. 215].

NOTE

This method should only be used if you are performing an integration with a requirements management tool. See Requirements Management Classes [p. 197].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
CreateRMTree(RMIntegrationValue rmv)
```

GetCSVErrorLog

Gets the CSV Error Log associated with the RequirementFolder [p. 138].

Type: `System.Collections.Generic.List<Compuware.QM.QADirector.SDK.CSVError>`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
GetCSVErrorLog()
```

InsertNewRequirement

Inserts a new RequirementNode [p. 146] into the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.RequirementNode`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
InsertNewRequirement()
```

ResetAndDeleteRMIntegration

This method resets the integration with the associated Requirement Management application and deletes all imported requirements from the Requirement Center in *QA*Director for the specified RMIntegrationValue [p. 215].

NOTE

This method should only be used if you are performing an integration with a requirements management tool. See Requirements Management Classes [p. 197].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
ResetAndDeleteRMIntegration(RMIntegrationValue rmv)
```

ResetRMIntegration

This method resets the integration with the associated Requirements Management application for the specified RMIntegrationValue [p. 215].

NOTE

This method should only be used if you are performing an integration with a requirements management tool. See Requirements Management Classes [p. 197].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
ResetRMIntegration(RMIntegrationValue rmv)
```

RMNewEP

This method creates and returns an execution plan with the specified name for the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.ExecutionPlan`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
RMNewEP(string EPName, bool bCreateGroups)
```

- `string EPName` is the name of the **`ExecutionPlan.`**
- `bool bCreateGroups` is used to create execution groups based on your requirement hierarchy. If false, the *Execution Plan* will contain a flat list of tests.

RMReplaceEP

Updates an existing **`ExecutionPlan`** by removing all the nodes in it and recreating the hierarchy from the RequirementFolder [p. 138].

Type: `Compuware.QM.QADirector.SDK.ExecutionPlan`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
RMReplaceEP(int EPDefnID, string EPName, bool bCreateGroups)
```

Parameters

- `int EPDefnID` is the ID of the **`ExecutionPlan`**.
- `string EPName` is the name of the **`ExecutionPlan.`**
- `bool bCreateGroups` is used to create execution groups based on your requirement hierarchy. If false, the *Execution Plan* will contain a flat list of tests.

UpdateRMTree

Updates all the requirements in a RequirementFolder [p. 138]. Accepts an RMIntegrationValue [p. 215] object and `RMUpdateOpts` object.

NOTE

This method should only be used if you are performing an integration with a requirements management tool. See Requirements Management Classes [p. 197].

Type: `bool`

Namespace: `Compuware.QM.QADirector.SDK.RequirementFolder`

Syntax

```
UpdateRMTree(RMIntegrationValue rmv,RMUpdateOpts upOpts)
```

Parameters

- `RMIntegrationValue rmv` - an RMIntegrationValue [p. 215] object.
- `RMUpdateOpts upOpts` - an RMUpdateOpts [p. 144] object.

RMIntegrationValue

Returns an `RMIntegrationValue` object.

Namespace: `Compuware.QM.QADirector.SDK`

Properties

Properties

CreatedByUser

Returns the User [p. 192] that created the RMIntegrationValue [p. 215].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.RMIntegrationValue`

ModifiedByUser

Returns the User [p. 192] that last modified the Client [p. 60].

Type: `Compuware.QM.QADirector.SDK.User`

Namespace: `Compuware.QM.QADirector.SDK.RMIntegrationValue`

RMDBKey

Gets the `RMDBKey` string for the RMIntegrationValue [p. 215].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.RMIntegrationValue`

RMID

Gets the `RMDBKey` numeric value for the RMIntegrationValue [p. 215].

Type: `int`

Namespace: `Compuware.QM.QADirector.SDK.RMIntegrationValue`

RMInfo1

Gets the `RMInfo1` string for the RMIntegrationValue [p. 215].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.RMIntegrationValue`

RMInfo2

Gets the `RMInfo2` string for the RMIntegrationValue [p. 215].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.RMIntegrationValue`

RMInfo3

Gets the `RMInfo3` string for the RMIntegrationValue [p. 215].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.RMIntegrationValue`

RMModule

Gets the `RMModule` string for the RMIntegrationValue [p. 215].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.RMIntegrationValue`

RMProject

Gets the `RMProject` string for the RMIntegrationValue [p. 215].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.RMIntegrationValue`

RMProjectID

Gets the `RMProjectID` value for the RMIntegrationValue [p. 215].

Type: `long`

Namespace: `Compuware.QM.QADirector.SDK.RMIntegrationValue`

RMToolName

Gets the `RMToolName` string for the RMIntegrationValue [p. 215].

Type: `string`

Namespace: `Compuware.QM.QADirector.SDK.RMIntegrationValue`

RMIntegrationValues

Returns a collection of `RMIntegrationValue` objects.

Namespace: `Compuware.QM.QADirector.SDK`

Methods

- GetRMIntegrationValue [p. 217]
- New [p. 217]

Methods

GetRMIntegrationValue

Gets the RMIntegrationValue [p. 215] object based on specified parameters.

Type: `Compuware.QM.QADirector.SDK.RMIntegrationValue`

Namespace: `Compuware.QM.QADirector.SDK.RMIntegrationValues`

Syntax

```
GetRMIntegrationValue(string sRMToolName,long iRMProjectID, string
sRMProjectName,string sRMInfo1, string sRMInfo2, string sRMInfo3)
```

New

Adds a new RMIntegrationValue [p. 215] object to the RMIntegrationValues [p. 217] collection based on the specified parameters.

Type: `Compuware.QM.QADirector.SDK.RMIntegrationValue`

Namespace: `Compuware.QM.QADirector.SDK.RMIntegrationValues`

Syntax

```
New(string sRMToolName, long iRMProjectID, string sRMProjectName, string
sRMDBKey, string sRMModule, string sRMInfo1, string sRMInfo2, string sRMInfo3)
```

Index

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Y