Mercury Quality Center Open Test Architecture Guide, Version 8.2 Service Pack 1

This manual, and the accompanying software and other documentation, is protected by U.S. and international copyright laws, and may be used only in accordance with the accompanying license agreement. Features of the software, and of other products and services of Mercury Interactive Corporation, may be covered by one or more of the following patents: United States: 5,511,185; 5,657,438; 5,701,139; 5,870,559; 5,958,008; 5,974,572; 6,137,782; 6,138,157; 6,144,962; 6,205,122; 6,237,006; 6,341,310; 6,360,332; 6,449,739; 6,470,383; 6,477,483; 6,549,944; 6,560,564; 6,564,342; 6,587,969; 6,631,408; 6,631,411; 6,633,912; 6,694,288; 6,738,813; 6,738,933; 6,754,701; 6,792,460 and 6,810,494. Australia: 763468 and 762554. Other patents pending. All rights reserved.

Mercury, Mercury Interactive, the Mercury logo, the Mercury Interactive logo, LoadRunner, WinRunner, SiteScope and TestDirector are trademarks of Mercury Interactive Corporation and may be registered in certain jurisdictions. The absence of a trademark from this list does not constitute a waiver of Mercury's intellectual property rights concerning that trademark.

All other company, brand and product names may be trademarks or registered trademarks of their respective holders. Mercury disclaims any responsibility for specifying which marks are owned by which companies or which organizations.

Mercury Interactive Corporation
379 North Whisman Road
Mountain View, CA 94043
Tel: (650) 603-5200
Toll Free: (800) TEST-911
Customer Support: (877) TEST-HLP
Fax: (650) 603-5300

© 2004 - 2005 Mercury Interactive Corporation, All rights reserved

If you have any comments or suggestions regarding this document, please send them via e-mail to documentation@mercury.com.

QCOTAG8.2/02
Table of Contents

Welcome to Quality Center Open Test Architecture ......................... v
Using This Guide ................................................................................ vi
Mercury Quality Center Documentation Set ....................................... vii
Online Resources ................................................................................ vii
Documentation Updates ....................................................................... ix
Typographical Conventions ................................................................. ix

Chapter 1: Integrating Custom Testing Tools ................................. 1
About Integrating Custom Testing Tools ............................................. 1
The Quality Center Open Test Architecture ...................................... 3
The TestType Mechanism ................................................................. 6

Chapter 2: Implementing Testing Tool Integration ....................... 9
About Implementing Testing Tool Integration .................................. 10
Creating Custom Test Types ............................................................. 11
TestType COM Class ........................................................................... 12
RemoteAgent DCOM Server .............................................................. 19
ScriptViewer ActiveX Control ........................................................... 31
ResultViewer ActiveX Control .......................................................... 39
ExecConfiguration ActiveX Control .................................................. 41
Registering Custom Test Types with Quality Center ..................... 43

Chapter 3: Using the Quality Center API ...................................... 53
About the Quality Center API .......................................................... 53
Integrating Your Applications with Quality Center ....................... 54
Accessing Quality Center API Functions ....................................... 55
Downloading the OTAClient80.dll ..................................................... 56
How the OTAClient80.dll Communicates with Quality Center ...... 56
Quality Center API Terminology ....................................................... 58
Table of Contents

Chapter 4: Quality Center Projects Data Structure ...........................59
Quality Center Projects ...................................................................59
Table Relationships .......................................................................60
Data Tables ....................................................................................62
System Tables and Security Tables ................................................89
Index..............................................................................................103
Welcome to Quality Center Open Test Architecture

Welcome to Mercury Quality Center (formerly TestDirector), Mercury Interactive’s Web-based test management tool. Quality Center helps you organize and manage all phases of the application testing process, including specifying testing requirements, planning tests, executing tests, and tracking defects.

Using Quality Center Open Test Architecture, you can integrate your own requirement and configuration management tools, defect tracking tools, third-party and custom tools, and modelling applications. You can:

➤ execute tests in your testing application on multiple hosts across a network, and analyze the test results from within the Quality Center environment.

➤ use the Quality Center COM-based API to enable your application to create, retrieve, and update Quality Center project records from within your test application environment.

Note: New API examples are continuously being added to the Knowledge Base on the Mercury Interactive Customer Support Web site. To search the Mercury Quality Center Knowledge Base for new information, select Using the Open Test Architecture - API from the Topic list.

For information about getting started with the Quality Center API, and for details of the classes, methods, and properties exposed, refer to the Quality Center Open Test Architecture API Reference.
Welcome

Using This Guide

The chapters in this guide are summarized below:

**Chapter 1  Integrating Custom Testing Tools**
Describes how the Quality Center API enables Quality Assurance testers to automate and extend the testing process by integrating custom or third-party testing tools with Quality Center.

**Chapter 2  Implementing Testing Tool Integration**
Describes how to create a custom test type. The custom test types you create with the Quality Center client can be accessed using the Quality Center user interface. You can use these test types the same way you use the built-in test types.

**Chapter 3  Using the Quality Center API**
Describes the API Architecture, including the relationships between the API component terminology and the Quality Center project terminology.

**Chapter 4  Quality Center Projects Data Structure**
Describes the project database, including the user tables and the security and system tables.
**Mercury Quality Center Documentation Set**

In addition to this guide, Quality Center comes with the following printed documentation:

**Mercury Quality Center User’s Guide** explains how to use Quality Center to organize and execute all phases of the testing process. It describes how to define requirements, plan tests, run tests, and track defects.

**Mercury Quality Center Administrator’s Guide** explains how to create and maintain projects using the Site Administrator, and how to customize projects using Project Customization.

**Mercury Quality Center Installation Guide** explains how to install Quality Center on a server machine in a cluster environment or as a stand-alone application.

**Mercury Quality Center Tutorial** is a self-paced guide teaching you how to use Quality Center to manage the software testing process.


**Online Resources**

Quality Center includes the following online resources:

---

**Note:** The Help button is located on the upper-right side of the Quality Center window.

---

**Readme** provides last-minute news and information about Quality Center.

**What’s New** describes the newest features in the latest versions of Quality Center. Click the Help button and choose What’s New.

**Books Online** displays the complete documentation set in PDF format. Online books can be read and printed using Adobe Reader which can be downloaded from the Adobe Web site (http://www.adobe.com). Click the Help button and choose Books Online.
Mercury Quality Center Online Help provides immediate answers to questions that arise as you work with Quality Center. It describes menu commands and dialog boxes, and shows you how to perform Quality Center tasks. Click the Help button and choose Online Help.

Mercury Quality Center Open Test Architecture API Reference provides a complete online reference for the Quality Center COM-based API. You can use the Quality Center open test architecture to integrate your own configuration management, defect tracking, and home-grown testing tools with a Quality Center project. Click the Help button and choose Books Online. Under Quality Center API, select Mercury Quality Center Open Test Architecture API Reference (Help file).

Mercury Quality Center Site Administrator Client API Reference provides a complete online reference for the Site Administrator Client COM-based API. You can use the Site Administrator Client API to enable your application to organize, manage, and maintain Quality Center users, projects, domains, connections, and site configuration parameters. Click the Help button and choose Books Online. Under Quality Center API, select Mercury Quality Center Site Administrator API Reference (Help file).

Customer Support Online uses your default Web browser to open the Mercury Interactive Customer Support Web site. This site enables you to browse the Mercury Support Knowledge Base and add your own articles. You can also post to and search user discussion forums, submit support requests, download patches and updated documentation, and more. The URL for this Web site is http://support.mercury.com. Alternatively, click the Help button and choose Customer Support Online.

Mercury Interactive on the Web uses your default Web browser to open Mercury Interactive's home page. This site provides the most up-to-date information on Mercury Interactive and its products. This includes new software releases, seminars and trade shows, customer support, educational services, and more. The URL for this Web site is http://www.mercury.com. Alternatively, click the Help button and choose Mercury Interactive on the Web.
Documentation Updates

Mercury Interactive is continuously updating its product documentation with new information. You can download the latest version of this document from the Customer Support Web site (http://support.mercury.com).

To download updated documentation:

1  In the Customer Support Web site, click the Documentation link.

2  Under Please Select Product, select TestDirector for Quality Center. If TestDirector for Quality Center does not appear in the list, you must add it to your customer profile. Click My Account to update your profile.

3  Click Retrieve. The Documentation page opens and lists the documentation available for the current release and for previous releases. If a document was recently updated, Updated appears next to the document name.

4  Click a document link to download the documentation.

Typographical Conventions

This book uses the following typographical conventions:

1, 2, 3  Bold numbers indicate steps in a procedure.
➤  Bullets indicate options and features.
>  The greater than sign separates menu levels (for example, File > Open).

Stone Sans  The Stone Sans font indicates names of interface elements (for example, “Click the Run button.”).

Bold  Bold text indicates function and object names.

Italics  Italic text indicates property and parameter names.

Arial  The Arial font is used for examples and statements that are to be typed in literally.

Courier New  The Courier New font is used for syntax examples in the object reference.

Courier New italic  The Courier New italic font is used for comments within examples.
Welcome
1

Integrating Custom Testing Tools

You can integrate your custom and third-party testing tools with Quality Center to create tests using these tools. You can then use Quality Center to configure the testing tools you are using, view test scripts created, execute the tests across multiple remote hosts, and view test results.

This chapter describes the architecture that enables Quality Center to integrate with custom and third-party testing tools. For information on implementing this integration, see Chapter 2, “Implementing Testing Tool Integration”.

This chapter describes:
➤ The Quality Center Open Test Architecture
➤ The Test Types Mechanism

About Integrating Custom Testing Tools

Quality Center enables you to integrate your custom and third-party testing tools so that you can continue to develop and use your current testing solution.

After you define your testing requirements and creating a test plan tree in Quality Center, you can create tests with Quality Center or other testing tools integrated with Quality Center. Once you have created tests, you define test sets in the Test Lab module. A test set is a group of tests designed to meet a specific testing goal. For example, to verify that the application being tested is functional and stable, you could create a sanity test set that checks the application’s basic features. You can then create other test sets to test the advanced features.
Additionally, you define the conditions that cause each test in the test set to be run. These can be a date and time (time condition), or a condition based on the result of another test (run condition). A typical run condition configures a test to run after another test in the test set has run and passed. By using this condition for all the tests in a test set, the tester can create a batch test in which the tests run in sequence until one of the tests fails, at which point the sequence is terminated.

After you design your tests and test sets, you can use Quality Center to execute tests created with third-party testing tools on multiple hosts across a network. After test execution is complete, you can use Quality Center reports and graphs to analyze test results. You can use customized controls to view the test script and results in a format compatible with the testing tool you are using. You can also add defects discovered during test execution to the Quality Center defect database, and track these defects until they are repaired. For a complete discussion of the Quality Center features and user interface, refer to the *Mercury Quality Center User’s Guide*. 
The Quality Center Open Test Architecture

The following diagram illustrates the Quality Center open test architecture.
The Quality Center physical environment consists of three types of entities:

- The Quality Center server
- The Quality Center client
- The testing application host

Note that two, or even all, of these entities can reside on one machine.

The server hosts the server modules and the Quality Center database. (The physical location of the database may vary, but it is included on the server for simplicity.)

The testing tool host executes the testing application and a remote agent module, that enables the client modules on the client machine to control the testing application and obtain the testing status. The testing application then uses the Quality Center API (Application Program Interface) to update the Quality Center server database with information regarding the test being run and the results. The Quality Center API is COM (Component Object Model)-based, and can be implemented directly through the testing application as an extension to the remote agent module, or through any other custom component.

The client hosts optional controls that enable the client user to view and set the testing application properties in its own custom format. The client machine and the server can be located anywhere on the Internet.

The Quality Center client uses custom controls to access testing details—such as the test script and test parameters—from the Quality Center server. The custom controls are needed to view the testing details in the custom format used by the specific testing application. These components are optional and are downloaded to the client from the server through the mechanism described in “Registering Custom Test Types with Quality Center” on page 43.
Quality Center uses the testing host remote agent to communicate with testing tools. By using Microsoft’s DCOM (Distributed Component Object Model) protocol, the remote agent is accessible directly over a network, enabling the testing tool to receive commands from the Quality Center client. For your testing application to receive commands from Quality Center, you must install a remote agent on each host you use for running tests. Note that the Quality Center client and testing host machines must be on the same LAN (Local Area Network).

When the Quality Center client requests to run a test, it first checks with the appropriate remote agent module that the testing application is ready. The client then sets the test parameters for the requested test through the remote agent and commands the agent to run the test. Once the test is run, the client can query the remote agent for the execution progress and results (for example, success or failure). The testing application can then communicate with the Quality Center server through the Quality Center API interfaces to update the server’s database with the test details and results.

The ability to run a test through the client on a remote testing host (running a custom or third-party testing application) and obtain the test’s results from the server enables you to use the custom or third-party testing tool as you would use integrated testing applications.
Chapter 1 • Integrating Custom Testing Tools

The TestType Mechanism

Interfacing your testing tool with Quality Center requires the Quality Center client to be aware of your custom or third-party testing tool. This is achieved through the TestType mechanism. The two major components of this mechanism are detailed below:

➤ The test_type.ini File
➤ The TestType Class

The following diagram illustrates the TestType access mechanism.
Chapter 1 • Integrating Custom Testing Tools

The test_type.ini File

The test_type.ini file on the client is downloaded from the archive file qcbin.war, found on the application server machine in the Quality Center virtual directory under Quality Center\application. The file contains a list of all the testing tools currently supported by the Quality Center server, with their associated TestType COM (Component Object Model) class ID. For more information on the TestType class, see “TestType COM Class” on page 12.

When you open the test_type.ini file, you can see the following:

[WR-AUTOMATED]
CLSID={E1ED35C0-8482-11D2-9399-0080C837F11F}

[VAPI-TEST]
CLSID={6D3B8D58-B5F5-11D2-9399-0080C837F11F}

[LR-SCENARIO]
CLSID={7B1A7474-AFAD-11D2-9399-0080C837F11F}

[DB-TEST]
CLSID={04D794C0-B9FC-11D2-9399-0080C837F11F}

[MANUAL]
CLSID={11A7DB23-A1A7-11D3-9CA4-0080C837F11F}

The file lists the supported testing applications such as WinRunner (WR-AUTOMATED), followed by their TestType class IDs (CLSID).

For the Quality Center client to support your custom or third-party testing application (through the modules mentioned in “The Quality Center Open Test Architecture” on page 3), you must add an entry to this file specifying your test type name, followed by its TestType class ID.

The test_type.ini file is discussed in greater detail in “Registering Custom Test Types with Quality Center” on page 43.
The TestType Class

If you know the class ID of the TestType class associated with your custom or third-party testing tool, you can access this class through COM. The TestType class contains general testing tool properties—such as the testing tool’s name and icon—as well as the class IDs for the testing tool’s remote agent and custom controls. These class IDs enable Quality Center to access the remote agent modules of the testing tools, and to use the appropriate custom controls to view the tests’ properties.

For more information on the TestType class, see “TestType COM Class” on page 12
Implementing Testing Tool Integration

To integrate custom and third-party testing tools with Quality Center, create a DCOM server and test type add-in for each testing tool, and register the add-in with Quality Center. The DCOM server enables Quality Center to interface with the testing tool on a remote testing host. The test type add-in consists of one or several OCX files that contain information enabling Quality Center to interface with the third-party testing tool information in the Quality Center database.

This chapter describes:

➤ Creating Custom Test Types
➤ TestType COM Class
➤ RemoteAgent DCOM Server
➤ ScriptViewer ActiveX Control
➤ ResultViewer ActiveX Control
➤ ExecConfiguration ActiveX Control
➤ Registering Custom Test Types with Quality Center
Chapter 2 • Implementing Testing Tool Integration

About Implementing Testing Tool Integration

For Quality Center to run tests and display test scripts and execution results created with a custom testing tool, Quality Center must use a remote agent and test type for the tool. The remote agent is a DCOM server residing on the remote testing host, interfacing with the testing tool. A test type is one or more COM components residing on the client, containing information that tells Quality Center how to interface with the data of the selected testing tool. A testing tool can support more than one test type.

You register the test type with Quality Center by specifying the class ID of the test type class in the Quality Center ini file. For more information, see “Registering Custom Test Types with Quality Center” on page 43. This identifies the main COM class of the test type. Quality Center creates an object of this class and uses its methods to interface with the testing tool through the remote agent and the custom ActiveX controls (DLL and OCX files).

For example, when Quality Center runs a WinRunner test, the WinRunner test type notifies Quality Center of the WinRunner remote agent’s class ID.

Quality Center supports two kinds of test types:

➤ **Pre-defined test types**: These are the default test types included with Quality Center. These enable you to execute various Mercury testing tools, such as running WinRunner automated tests on remote hosts, viewing test scripts, and displaying test results.

➤ **Custom add-in test types**: You create this kind of test type to run tests that were created with other testing tools on remote hosts, and view test scripts and results. A custom test type must contain one or more of the components described in “Creating Custom Test Types” on page 11.
Creating Custom Test Types

Custom test types enable you to execute tests that were created with your own testing tools on remote hosts, and view test scripts and results. Each custom test type may include the following components:

- **TestType COM Class**: Contains the test type general properties. This component is mandatory.
- **Remote Agent DCOM Server**: Runs tests locally and remotely.
- **ScriptViewer ActiveX Control**: Shows the test script. This component is optional.
- **ResultViewer ActiveX Control**: Shows the test results. This component is optional.
- **ExecConfiguration ActiveX Control**: Configures the testing tool. This component is optional.

To use a custom test type, Quality Center creates an object of the TestType COM class, and uses its methods to obtain the user interface properties of the test type, to create the test script template, and to obtain the class ID of the ScriptViewer, ResultViewer, ExecConfiguration ActiveX controls, and RemoteAgent DCOM server.

**Note**: The type library required for creating the test type is downloaded and registered automatically when you log on to Quality Center from the workstation.
The **TestType** object provides the interface between the testing tool and Quality Center. The **Init** method of the **TestType** object gets an **IDispatch** reference to a **TDConnection** object, used to access the Quality Center database. For more information about the **TDConnection** object, see *Quality Center Open Test Architecture API Reference*.

The following are the properties and methods that must be implemented by the **TestType** object.
## TestType Properties

### Simple Data Type Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>R/W</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanCreateScriptTemplate</td>
<td>R</td>
<td>Long</td>
<td>Indicates whether the test type supports creating script templates. Returns 0 if the test cannot be created; 1 if the test can be created; 2 if the test supports creating script templates.</td>
</tr>
<tr>
<td>ExecConfigCLSID</td>
<td>R</td>
<td>String</td>
<td>The class ID of the ExecConfiguration class associated with this test type, in the following format: <code>{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}</code>, where X is a hexadecimal character. If ExecConfiguration is not supported, this property returns an empty string.</td>
</tr>
<tr>
<td>LastErrorMessage</td>
<td>R</td>
<td>String</td>
<td>The most recent error message.</td>
</tr>
<tr>
<td>RemoteAgentCLSID</td>
<td>R</td>
<td>String</td>
<td>The class ID of the RemoteAgent class associated with this test type, in the following format: <code>{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}</code>, where X is a hexadecimal character. If RemoteAgent is not supported, this property returns an empty string.</td>
</tr>
</tbody>
</table>
**Chapter 2 • Implementing Testing Tool Integration**

### TestType Methods

#### Init Method

Initializes the TestType object.

**Syntax**

```
HRESULT Init(VARIANT TDConnection)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDConnection</td>
<td>Required. A variant containing the IDispatch reference to the TDConnection object. For a description of the TDConnection object, see the TDConnection object description in Quality Center Open Test Architecture API Reference.</td>
<td>None</td>
</tr>
</tbody>
</table>

CreateScriptTemplate Method

Creates a script template for the specified test. The test type creates a script on the local client machine, and uploads it to the server side test repository.

Syntax

HRESULT CreateScriptTemplate( long TestKey, BSTR LocalPath, long Value )
Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestKey</td>
<td>Required. The key of the test for which the template is created.</td>
<td>None</td>
</tr>
</tbody>
</table>
| LocalPath      | Input - the path on the local host in which the client wishes to place the script (the path does not have to exist). Output - one of the following two options:  
- If the script is created and loaded correctly by the TestType object, and the database has been updated, the return value is an empty string.  
- If the upload and database update is performed by Quality Center rather than by the TestType object, the return value is the path in which the script was created and from which it is downloaded (it does not have to be the same path as the input). |
| Value          | Returns TRUE (1) if the call is successful, FALSE (0) otherwise.            | None    |

Returns

In Visual Basic, *Value* is the return value, and is not passed as an argument. In C++, *Value* is passed as an argument, and the HRESULT is returned.
TestType Example

The following example is of the main TestType class. The example is developed step by step with the other remote execution component examples, so that executing all the steps included in this section results in a (simple) working remote test execution mechanism. The example uses Microsoft Visual Basic 6.0.

Using Microsoft Visual Basic, create a new ActiveX Exe project. Declare the following class members in the code window (global declaration area):

```vbnet
Public RemoteAgentCLSID As String ' Run Remote Agent Class ID
Public ScriptViewerCLSID As String ' ActiveX script viewer Class ID
Public TestingToolName As String ' Remote Agent name
Public ExecConfigCLSID As String ' Configuration Class ID
Public CanCreateScriptTemplate As Long ' Flag
Public LastErrorMessage As String ' Most common error message
Public ResultViewerCLSID As String ' ActiveX result viewer Class ID
```

These members represent the main TestType object properties. Add the following sub-routine to the code:

```vbnet
Private Sub Class_Initialize()
' This sub-routine is run when the TestType is loaded.
' Note that at this time, none of the optional controls is supported, as their class ID strings are empty.
' Also note that the remote agent class ID is empty, as no such agent yet exists.
' This example does not support script templates, and has only one error type.

    ExecConfigCLSID = "" ' No configuration utility provided
    RemoteAgentCLSID = "[Class ID for the Run Remote Agent]"
    ScriptViewerCLSID = "[Class ID for ActiveX script viewer]"
    TestingToolName = "Script"
    CanCreateScriptTemplate = 1 ' create new test feature turned on
    LastErrorMessage = "" ' Testing issues
    ResultViewerCLSID = "" ' No result viewer ActiveX provided
```

17
Chapter 2 • Implementing Testing Tool Integration

    LoadPicture ""
End Sub

Public Sub Init(mytd As Variant)
    ' You may need a TDConnection to use OTA functions in your application.
    ' If you do, add a project reference to the OTA:
    ' In Projects > References, check the "OTA COM 8.0 Type Library."
    ' and declare a module-level variable: Private td As TDConnection
    Set td = mytd 'Get the td object to be used if needed
End Sub

Public Function CreateScriptTemplate(TestKey As Long, _
    ByRef LocalPath As String) As Long
    ' Create a batch file to emulate a test script for
    ' this example
    Dim myf As String
    LocalPath = "c:\temp"
    myf = "C:\temp\batch.bat"
    Open myf For Output As #1
    Print #1, "ping localhost" & vbCrLf & "pause" 'create script template
    Close #1
    CreateScriptTemplate = 1
End Function

Public Function GetBitmap(Status As Long) As Long
    GetBitmap = 0 'no bitmap
End Function

In the class properties window, change the Instancing property to 6 - GlobalMultiUse.

Choose Project > Properties. Click the General tab. Change the project’s name to MyTestType. Select Unattended Execution and Upgrade ActiveX Controls. In the same tab, in the Threading Model section, select the Thread Per Object option. Save the project as MyTestType.
Choose File > Make MyTestType.exe to compile the class and register it.

Note that this example is also incorporated in the other examples in this chapter.

**RemoteAgent DCOM Server**

Quality Center communicates with other testing tools via a remote agent. The remote agent resides on the host machine with the testing tool, and uses a DCOM protocol to communicate over the network with the host machine on which Quality Center resides.

For the remote agent to communicate with Quality Center, the agent must implement the `IDispatch` interface or dual interfaces, as well as the following methods.

**is_host_ready Method**

Before test execution, the Quality Center client uses this method to ask the remote agent to ensure that the designated testing host is available and ready to run the test.

**Syntax**

```csharp
HRESULT is_host_ready( BSTR descr )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>descr</td>
<td>Required. A string for the description of the reason the host is not ready.</td>
<td>None</td>
</tr>
</tbody>
</table>

**Return Value**

Returns S_OK if the testing tool and host are ready to run tests, and S_FALSE otherwise.
set_value Method

Before the testing tool can run a test, it must receive information from Quality Center, including the name of the test, the database to which the test belongs, and other test-related information. This information enables the testing tool to retrieve data from the Quality Center database, run tests, and return the test results to Quality Center.

This method accepts two parameters: the parameter name to be set and its designated value.

Syntax

HRESULT set_value (BSTR prm_name, BSTR prm_value)

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>prm_name</td>
<td>Required. The name of the parameter to be set. Case insensitive. This method supports the parameter names described in the table below.</td>
<td>None</td>
</tr>
<tr>
<td>prm_value</td>
<td>Required. The value to be set. Case insensitive.</td>
<td>None</td>
</tr>
</tbody>
</table>

Supported Parameter Names

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database_name</td>
<td>The name of the active Quality Center database.</td>
</tr>
<tr>
<td>domain_name</td>
<td>The name of the Quality Center domain in which test and result information is stored.</td>
</tr>
<tr>
<td>host_name</td>
<td>The name of the host on which the remote agent test type is run.</td>
</tr>
<tr>
<td>password</td>
<td>The user's password.</td>
</tr>
<tr>
<td>plan_status</td>
<td>The planning mode status of the test.</td>
</tr>
<tr>
<td>project_name</td>
<td>The name of the Quality Center project in which test and result information is stored.</td>
</tr>
</tbody>
</table>
responsible

The name of the user responsible for the project.

runner_result_dir

The name of the test run results directory.

scheduler_version

The software version number of the scheduler.

subject

The subject folder to which the test belongs in the test plan tree.

sys_computer_name

The name of the PC on which the Quality Center client is running.

sys_user_name

The login user name for the user logged in on the Quality Center client PC.

TDAPI_host_name

The name of the host on which the Quality Center server is running.

test_id

The ID of the test to be run.

test_name

The name of the test to be run.

test_path

The full path of the test to be run.

test_set

A string of form: \{test_set: "<test set path in the test tree>" testcycle_id: "148~1"\}

test_set_id

The ID of the test set to which the test belongs.

test_set_user1...99

The value of a user field in the Test in the Test Set table.

test_type

The custom test type.

test_user1...99

The value of a user field in the Tests table.

test_instance

The ID of the test instance inside a test set.

testcycle_id

The ID of the test.

tester_name

The name of the tester assigned to run the test.

ttest_name

The name of the test to be run with a “[1]” instance prefix.
user_name

The name of the user running the test.

**Return Value**

Returns S_OK if the call succeeds, and S_FALSE otherwise.

**run Method**

The `run` method instructs the testing tool to load and run the test. This function also launches the testing tool if it is not already running.

**Syntax**

```c
HRESULT run()
```

**Return Value**

Returns S_OK if the call succeeds, and S_FALSE otherwise.

**tdstop Method**

The `tdstop` method instructs the testing tool to terminate the test that is currently running.

**Syntax**

```c
HRESULT tdstop()
```

**Return Value**

Returns S_OK if the call succeeds, and S_FALSE otherwise.

**get_status Method**

During test execution, Quality Center checks the status of the testing tool and displays this information. This enables the tester to monitor the test’s progress at each stage of the run.

This function returns the current status of the testing tool.

**Syntax**

```c
HRESULT get_status (BSTR descr, BSTR status)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>descr</td>
<td>Required. Returns a verbal description of the testing tool's current status.</td>
<td>None</td>
</tr>
<tr>
<td>status</td>
<td>Returns one of the following values as a generic description of the current testing tool status.</td>
<td>None</td>
</tr>
</tbody>
</table>

Supported Status Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSY</td>
<td>The testing tool is currently running another test.</td>
</tr>
<tr>
<td>END_OF_TEST</td>
<td>The testing tool has reached the end of the current test.</td>
</tr>
<tr>
<td>FAILED</td>
<td>The testing tool has failed.</td>
</tr>
<tr>
<td>INIT</td>
<td>The testing tool is in its initialization stage.</td>
</tr>
<tr>
<td>LOGICAL_RUNNING</td>
<td>The testing tool is running the test.</td>
</tr>
<tr>
<td>PAUSED</td>
<td>The testing tool has paused execution of the current test.</td>
</tr>
<tr>
<td>READY</td>
<td>The testing tool is ready to run the test.</td>
</tr>
<tr>
<td>STOPPED</td>
<td>The testing tool has stopped execution of the current test.</td>
</tr>
<tr>
<td>TEST_PASSED</td>
<td>The test has been successfully completed.</td>
</tr>
<tr>
<td>TEST FAILED</td>
<td>The test failed.</td>
</tr>
<tr>
<td>RETRY</td>
<td>You cannot execute the test on the current host. Try to execute the test on another host from the attached host group.</td>
</tr>
</tbody>
</table>

Return Value

Returns S_OK if the testing tool and host are ready to run tests, and S_FALSE otherwise.
RemoteAgent Example

The following is an example of the remote agent class that runs the test.

Using Microsoft Visual Basic, create a new ActiveX EXE project. In Projects > References, check the "OTA COM 8.0 Type Library."

Declare the following class members in the code window (global declaration area):

' These variables are required by the interface:
Private Status As String ' The current testing status
Private Descr As String ' The current testing description
Private ServerName As String ' The server URL
Private ProjectName As String ' The current project's name
Private DomainName As String ' The Quality Center domain name
Private UserName As String ' The current user name
Private SysUserName As String ' The workstation system user
Private Password As String ' The current user's password
Private TestPath As String ' The current test's path
Private TestName As String ' The current test's name
Private TestSet As String ' The current test set's name
Private TestID As Integer ' The current test's ID
Private TestInst As Long ' The test instance
Private TestIDInst As String ' The test instance descriptor
Private S_OK As Long ' Return value if the run is OK
Private S_FALSE As Long ' Return value if the run failed
Private END_OF_TEST As Long ' Return value for end of test

' These variables are not required by the interface.
' They are for a log file for your custom messages
' Use requires a project reference to
' Microsoft Scripting Runtime
Private LogFile As Scripting.TextStream
Private oSFile As Scripting.File
Private oFileSys As Scripting.FileSystemObject

Add the class initialization sub-routine:

Private Sub Class_Initialize()
' Initialization method of some class members.
S_OK = 0
S_FALSE = 1
END_OF_TEST = 4

On Error GoTo initerr
Set oFileSys = _
    CreateObject("Scripting.FileSystemObject")
Set LogFile = _
    oFileSys.OpenTextFile _
        ("D:\Temp\MyOTARunAgent.log", ForAppending)
LogFile.WriteLine "Initialize " & CStr(Now)
initerr:
End Sub

This sub-routine initializes the return value variables. Add the following RemoteAgent interface methods:

Public Function is_host_ready(Descr As String) As Long
    ' Quality Center calls this method to check if the
    ' host is ready
    ' In this example, is_host_ready always reports that
    ' the host is ready, and changes the description to
    ' "Ready"
    Descr = "Ready"
    is_host_ready = S_OK
End Function

Public Function set_value(ByVal name As String, _
    ByVal Value As String) As Long
    ' Quality Center will use this method to set the
    ' variables declared.
    Select Case name
    Case "TDAPI_host_name"
        ServerName = Value
    Case "project_name"
        ProjectName = Value
End Function
Case "domain_name"
  DomainName = Value
Case "user_name"
  UserName = Value
Case "sys_user_name"
  SysUserName = Value
Case "password"
  Password = Value
Case "test_name"
  TestName = Value
Case "test_path"
  TestPath = Value
Case "test_set_id"
  TestSet = Value
Case "test_set"
  TestIDInst = Value
Case "test_id"
  TestID = Val(Value)
End Select

set_value = S_OK

On Error Resume Next
  LogFile.WriteLine "set_value: "
  & " name = " & name & ", value = " & Value
End Function

Public Function get_status(StatusDescription As String, 
  CurrentStatus As String) As Long
' Quality Center calls this method to check the
' status and description of the run.
  StatusDescription = Descr
  CurrentStatus = Status
  get_status = S_OK
End Function

Public Function run() As Long
' Quality Center calls this method to run the test
' and update the test status and description.
' In this example, batch.bat is downloaded
' and run.

OnError GoTo runErr

LogFile.WriteBankLines 3
LogFile.WriteLine "Start run " & CStr(Now)

Dim td As New TDAPIOLELib.TDConnection
Dim tsfact As TestSetFactory
Dim ts As TestSet
Dim tsfact As TSTestFactory
Dim tst As TSTest
Dim rfact As RunFactory
Dim theRun As TDAPIOLELib.run

' These variables are needed to get
' batch.bat for this example:
Dim tfact As TDAPIOLELib.TestFactory
Dim theTest As TDAPIOLELib.Test
Dim LocalScriptPath As String
Dim ExtStorage As TDAPIOLELib.ExtendedStorage

' Connect to the project
    td.InitConnectionEx ServerName
    td.ConnectProjectEx DomainName, ProjectName, UserName, Password

' Get a local copy of batch.bat
' Get the test
    Set tfact = td.TestFactory
    Set theTest = tfact.Item(TestID)
    LogFile.WriteLine "The test name = " & theTest.name

' Get the file
    Set ExtStorage = theTest.ExtendedStorage
LocalScriptPath = ExtStorage.Load("-r batch.bat", True)

' Add the file name to the path returned from Load
' so it directly refers to the file.
LocalScriptPath = LocalScriptPath & ";batch.bat"
LogFile.WriteLine "LocalScriptPath = " & LocalScriptPath

' Update the status variables for use in get_status
Status = "RUNNING"
Descr = "Running..."

' Run the application and update the status
Dim rc
rc = Shell(LocalScriptPath)
Status = "END_OF_TEST"
Descr = "Completed"

' Record the run in the project
Set tsfact = td.TestSetFactory
Set ts = tsfact(TestSet)
Set tstfact = ts.TSTestFactory

Dim ttt As String

'Get the TSTest id:
' TestID & "~" & the instance of this test
' in this test set. For example, "84~2"
'TestIDInst is the test set, for example:
' { test_set: "Root\Sample Tests\My Sample Tests",
  testcycle_id: "84~1"
'}
ttt = Mid(TestIDInst, _
              InStr(1, TestIDInst, "testcycle_id: ") + 15)
'Get the instance number
ttt = Mid(ttt, InStr(1, ttt, "~") + 1)
TestInst = Val(Mid(ttt, 1, InStr(1, ttt, "~")))
'Get the TSTest object using the TestID and Instance
Set tst = tstfact(TestID & "~" & TestInst)
'Get the run factory for the TSTest
Set rfact = tst.RunFactory

'Set the run data
Dim data(0 To 1)
'Test run name (ID)
data(0) = "New Run 7"
'User who ran the test (from set_value)
data(1) = UserName

' Create the new Run
Set theRun = rfact.AddItem(data)
'Mark the run Passed
theRun.Status = "Passed"
theRun.Post

run = S_OK
Exit Function

runErr:
Dim msg$
   On Error Resume Next
   msg = "Error in run: " & Err.Description
   LogFile.WriteLine msg
   On Error GoTo runErr
   Resume Next
End Function

'Close the log file at the end.
Private Sub Class_Terminate()
   LogFile.Close
End Sub

In the class properties window, change the **Instancing** property to 6 - GlobalMultiUse.
Choose **Project > Properties** and click the **General** tab. Change the project’s name to MyRunAgent. Select **Unattended Execution** and **Upgrade ActiveX Controls**. In the same tab, in the **Threading Model** section, select the **Thread Per Object** option.

Click the **Component** tab and select **Remote Server Files**.

Save the project as MyRunAgent.

Choose **File > Make MyRunAgent.exe** to compile the class and register it.

The remote agent is now ready to run. For information on registering and running the remote agent with Quality Center, see “Registering Custom Test Types Example” on page 46.
ScriptViewer ActiveX Control

Each custom test type can include a ScriptViewer ActiveX control, enabling you to view the stored scripts in the test repository. This control can be activated within an ActiveX hosting frame. The ScriptViewer control must implement the following methods.

ScriptViewer Properties

Simple Data Type Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>R/W</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScriptReadOnly</td>
<td>R/W</td>
<td>Boolean</td>
<td>Set by Quality Center to indicate that the script should not be altered. For example, it may be locked by a version control system.</td>
</tr>
</tbody>
</table>

TestType Methods

Init Method

Initializes the ScriptViewer object.

**Syntax**

```c
HRESULT Init( VARIANT TDConnection )
```
Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDConnection</td>
<td>Required. A variant containing the IDispatch reference to the TDConnection object. For a description of the TDConnection object, see the TDConnection object description in Quality Center Open Test Architecture API Reference.</td>
<td>None</td>
</tr>
</tbody>
</table>

ShowTest Method
Displays the specified test script in the Script Viewer control.

Syntax

```c
HRESULT ShowTest( long TestKey )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestKey</td>
<td>Required. The requested test’s ID.</td>
<td>None</td>
</tr>
</tbody>
</table>

SaveScript Method
Saves the test script to the Quality Center project.

Syntax

```c
HRESULT SaveScript( Variant Flags )
```
Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flags</td>
<td>Required. Always pass zero (0).</td>
<td>None</td>
</tr>
</tbody>
</table>

ScriptViewer Example

The following is an example of the script viewer control that allows you to view the test.

Using Microsoft Visual Basic, create a new “ActiveX Control” project. Choose Project > References, and select the OTA COM 8.0 Type Library reference. For this example, you must also select a reference to the Microsoft Scripting Runtime, though you may not need it for your production code.

Add a text box to the user control. The script contents are displayed inside the box. Add a Command button named cmdSave. The button is used to save changes to the script.

Declare the following class members in the code window (global declaration area):

```vbs
' A ScriptReadOnly Boolean variable is required
' by the interface
Public ScriptReadOnly As Boolean

'These variables you can name as you like
Private td As New TDAPIOLELib.TDConnection
Private tfact As TDAPIOLELib.TestFactory
Private theTest As TDAPIOLELib.Test
Private LocalScriptPath As String
Private ExtStorage As TDAPIOLELib.ExtendedStorage

'Use of these variable requires project reference to
'Microsoft Scripting Runtime
Dim ScriptFile As Scripting.TextStream
Dim oSFile As Scripting.File
Dim oFileSys As Scripting.FileSystemObject
```
Add the interface methods: Init, SaveScript, and ShowTest.

The **Init** method sets the `td` object to the `ITDConnection` object passed by the Quality Center client, and clears the list box.

```vba
Public Sub Init(mytd As Variant)
    ' Initialization method, Quality Center will call this
    ' method and pass an TDConnection class object.
    Set td = mytd
End Sub
```

The **ShowTest** method gets the path for the specified test using its key, reads the script file (Batch.bat) from this location, and displays it in the text box.

```vba
Public Sub ShowTest(TestKey As Long)
    ' ShowTest(TestKey As Long) is required by the interface
    ' Quality Center will call the ShowTest method and pass
    ' the TestKey.
    Dim mydata As String

    On Error GoTo ShowTestErr

    'Get the test
    Set tfact = td.TestFactory
    Set theTest = tfact.Item(TestKey)

    'Get a local copy of the script
    'In this example, the script is "batch.bat"
    Set ExtStorage = theTest.ExtendedStorage
    LocalScriptPath = ExtStorage.Load("-r batch.bat", True)

    'Add the file name to the path returned from Load
    'so it directly refers to the file.
    LocalScriptPath = LocalScriptPath & "\batch.bat"
End Sub
```
'Open the script
Set oFileSys = _
   CreateObject("Scripting.FileSystemObject")
Set ScriptFile = _
   oFileSys.OpenTextFile(LocalScriptPath, _
       ForReading, False)

'Get the contents of the file into a string variable
If Not ScriptFile.AtEndOfStream Then _
   mydata = ScriptFile.ReadLine
Do While ScriptFile.AtEndOfStream <> True
   'List1.AddItem ScriptFile.ReadLine
   mydata = mydata & vbCrLf & ScriptFile.ReadLine
Loop
ScriptFile.Close

' Load the text box with the script text,
' Note that the text box must have the Multi-line
' attribute enabled
Text1.Text = mydata
' This is the initial state, so:
Text1.DataChanged = False

' Enable or disable the text box and save button,
' and make the file R/O or R/W depending
' on the lock status.
Set oSFile = oFileSys.GetFile(LocalScriptPath)
If ScriptReadOnly Then
   oSFile.Attributes = 1 'ReadOnly
   MsgBox "File locked by version control system " _
      & "or by Quality Center." _
      & vbCrLf & "Changes made to file " _
      & "will not be saved"
   Text1.Enabled = False
   cmdSave.Visible = False
Else
   oSFile.Attributes = 0
   Text1.Enabled = True
Chapter 2 • Implementing Testing Tool Integration

```
cmdSave.Visible = True
End If

Exit Sub
ShowTestErr:
  On Error Resume Next
  Dim msg$
  msg = " on " & theTest.Name
  msg = "ShowTest error" & msg & vbCrLf
  msg = msg & Err.Description
  MsgBox msg
End Sub
```

The **SaveScript** method uploads changes to the script to the Quality Center project.

```
Sub SaveScript(Optional Flags As Variant = 0)
  'SaveScript(Flags) is required by the interface.
  'Quality Center may call this method directly.

  Dim FileArchive As Integer

  'If the file was never downloaded for editing, exit
  If Len(LocalScriptPath) = 0 Then Exit Sub

  'If the file is locked (probably by a
  ' configuration manager), exit
  If ScriptReadOnly Then Exit Sub

  ' If the file hasn't changed, exit
  Set oSFile = oFileSys.GetFile(LocalScriptPath)
  FileArchive = oSFile.Attributes And Archive
  'File not changed
  If FileArchive <> Archive Then Exit Sub

  ' Ensure that there's an ExtendedStorage object
```
Chapter 2 • Implementing Testing Tool Integration

If (ExtStorage Is Nothing) Then _
    Set ExtStorage = theTest.ExtendedStorage

    'Save the file to the Quality Center Project
    ExtStorage.Save "-r batch.bat", True

End Sub

For this example, the Save button click event writes the contents of the text box to the local file.

Private Sub cmdSave_Click()
    ' The cmdSave button ("Save") on the dialog box

    ' If the user didn't make any changes, exit
    If Not Text1.DataChanged Then Exit Sub

    'Delete the existing file
    oFileSys.DeleteFile LocalScriptPath, True

    'Create a new one and write the contents of the
    ' text box to the file
    Set ScriptFile = _
        oFileSys.CreateTextFile(LocalScriptPath, True)
    ScriptFile.Write Text1.Text
    ScriptFile.Close

    ' Set the no-change status for the text box.
    Text1.DataChanged = False

    ' Save the script to the Quality Center project
    SaveScript 0
End Sub

For this example, you can set ScriptReadOnly in your code to exercise the different options. In production code, Quality Center will set this value.
Chapter 2 • Implementing Testing Tool Integration

Private Sub UserControl1_Initialize()
    ScriptReadOnly = False
End Sub

Choose **Project** > **Properties**. Click the **General** tab, and rename the project **MyScriptViewer**. Save the project under this name.

Choose **File** > **Make MyScriptViewer.ocx** to compile the ActiveX control class and register it with Windows.

To see the ScriptViewer control at work, you need to register it with Quality Center. For more information, see “Registering Custom Test Types Example” on page 46.
ResultViewer ActiveX Control

A test type can include a ResultViewer ActiveX control. This control is created within an ActiveX hosting frame. The ResultViewer control must implement the following methods.

**Init Method**
Initializes the ResultViewer object.

**Syntax**

```c
HRESULT Init(VARIANT TDConnection)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDConnection</td>
<td>Required. A variant containing the IDispatch reference to the TDConnection object. For a description of the TDConnection object, see the TDConnection object description in Quality Center Open Test Architecture API Reference.</td>
<td>None</td>
</tr>
</tbody>
</table>

**ShowResultEx Method**
This method displays the result of the specified test instance in a test set in the Result Viewer. This method is an extension to the ShowResult method (see below), and supports multiple instances of a test in a test set.

**Syntax**

```c
HRESULT ShowResultEx(VARIANT TestSetKey,
                     VARIANT TSTestKey,
                     VARIANT ResultKey)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestKey</td>
<td>Required. The test’s key.</td>
<td>None</td>
</tr>
<tr>
<td>TestSetKey</td>
<td>Required. The test set’s key.</td>
<td>None</td>
</tr>
<tr>
<td>ResultKey</td>
<td>Required. The test run’s result key.</td>
<td>None</td>
</tr>
</tbody>
</table>

ShowResult Method

This is an outdated method that is only supported for backward compatibility. The method displays the result of the specified test in the Result Viewer. This method’s calling format does not support multiple instances of a specific test in one test set.

Syntax

```c
HRESULT ShowResult(  long TestKey,  
                     long TestSetKey,  
                     long ResultKey )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestKey</td>
<td>Required. The test’s key.</td>
<td>None</td>
</tr>
<tr>
<td>TestSetKey</td>
<td>Required. The test set’s key.</td>
<td>None</td>
</tr>
<tr>
<td>ResultKey</td>
<td>Required. The test run’s result key.</td>
<td>None</td>
</tr>
</tbody>
</table>
ExecConfiguration ActiveX Control

A test type can include an ExecConfiguration ActiveX Control. This control is used to configure tests and testing tools in the Quality Center execution mode. The control returns configuration strings (in the testing tool’s internal format) to the Quality Center client call, which saves them in the database. Test configuration information is saved per test in a test set. Test type configuration is saved per project.

When Quality Center activates an ExecConfiguration ActiveX Control, it sets the TestConfiguration and TestTypeConfiguration properties, then calls the ShowExecConfiguration method in the ExecConfiguration ActiveX Control. When the session ends, Quality Center gets the TestConfiguration and TestTypeConfiguration properties and saves them in the Quality Center database. When Quality Center executes a test, it uses the set_value function to send the configuration strings to the appropriate remote agent. For a description of the set_value function, see “set_value Method” on page 20.

The ExecConfiguration ActiveX Control is created within an ActiveX hosting frame. The ExecConfiguration control must implement the following properties and methods.

ExecConfiguration ActiveX Control Properties

Simple Data Type Properties:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>R/W</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestConfiguration</td>
<td>R/W</td>
<td>String</td>
<td>Returns or sets the configuration strings for the test specified by the ShowExecConfiguration or ShowExecConfigurationEx methods.</td>
</tr>
<tr>
<td>TestTypeConfiguration</td>
<td>R/W</td>
<td>String</td>
<td>Returns or sets the general test type configuration string.</td>
</tr>
</tbody>
</table>
Chapter 2 • Implementing Testing Tool Integration

**ExecConfiguration ActiveX Control Methods**

**Init Method**
Initializes the ExecConfiguration object.

**Syntax**

```c
HRESULT Init(VARIANT TDConnection)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDConnection</td>
<td>Required. A variant containing the IDispatch reference to the TDConnection object. For a description of the TDConnection object, see the TDConnection object description in Quality Center Open Test Architecture API Reference.</td>
<td>None</td>
</tr>
</tbody>
</table>

**ShowExecConfigurationEX Method**
This method displays the configuration of the specified test in the specified test set. The method is an extension to the ExecConfiguration method (see below), and supports multiple instances of a test in a test set.

**Syntax**

```c
HRESULT ShowExecConfigurationEx(VARIANT TestSetKey,
                               VARIANT TSTestKey)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestSetKey</td>
<td>Required. The test set’s key.</td>
<td>None</td>
</tr>
<tr>
<td>TSTestKey</td>
<td>Required. The test’s key in the test set.</td>
<td>None</td>
</tr>
</tbody>
</table>
**ShowExecConfiguration Method**

This is an outdated method that is supported for backward compatibility. The method displays the configuration of the specified test in the specified test set. This method’s calling format does not support multiple instances of a specific test in one test set.

**Syntax**

```c
HRESULT ShowExecConfiguration( long TestKey,
                                long TestSetKey )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestKey</td>
<td>Required. The test’s key.</td>
<td>None</td>
</tr>
<tr>
<td>TestSetKey</td>
<td>Required. The test set's key.</td>
<td>None</td>
</tr>
</tbody>
</table>

**Registering Custom Test Types with Quality Center**

To use a custom test type, you need to register it with Quality Center.

**To register a custom test type with Quality Center:**

1. Go to the *application* directory under the Quality Center virtual directory on the application server machine.

2. Create a temporary scratch directory. Copy qcbin.war to the scratch directory and extract the files.

3. Copy the test type files (the class and controls file you created) into the *Install* directory under your scratch directory. Change the files’ extensions—for example, change *.exe to *.xxx, *.dll to *.lld, *.ocx to *.xco, and so forth.
Chapter 2 • Implementing Testing Tool Integration

4 Add a section to `<scratch>\Install\test_type.ini` with the following format:

```
[TEST TYPE NAME]
CLSID={XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

where [TEST TYPE NAME] is the name of the test type, and X is a hexadecimal character of the class ID for the TestType class.

For example:

```
[VAPI-TEST]
CLSID={6D3B8D58-B5F5-11D2-9399-0080C837F11F}
```

This file will be downloaded to each client machine, and enables Quality Center to recognize the test type and its controls. Note that on existing clients, you must delete `test_type.ini` from the client so that the client downloads the revised version of this file. In the Common Data section of the file `setup_a_80mp.ini`, note the default client root directory for the file download. It is in the Destination item line, and is generally:

```
Destination=%CommonDir%\Mercury Interactive\TD2000_80
```

which evaluates to C:\Program Files\Common\Mercury Interactive\TD2000_80

5 Add a section to `<scratch>\setup_a_80mp.ini` with the following format for each of the files to be downloaded to client machines (that is, the files you added to the Install directory, excluding the test_type.ini file):

```
[file_name]
URLName=<URLName>
Shortname=<shortname>
Description=<description>
Register=<register>
ProgID=<progID>
Version=<version>
Checksize=<checksize>
```
The required parameters within the setup ini file are described below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>URLName</strong></td>
<td>Required. The location of the source file to be downloaded. Must start with %URL% and continue with the relative location under the main IIS virtual directory. For example: URLName=%URL%\Install\MyFile.xco</td>
</tr>
</tbody>
</table>
| **shortname** | The options are:  
- The file name - the file is downloaded to the default destination directory.  
- The file name with the full path - the file is downloaded to the specified directory.  
- The file name with a partial path - the file is downloaded to the specified subdirectory of the default directory.  
Note that the file name extension here must be the runtime extension. For example: MyFile.ocx |
| **description** | Optional. A free text description that is displayed during file download. For example: Description=My Test Type Manager |
| **register** | Optional. Relevant only for COM/DCOM servers (EXE,OCX,DLL). There are two possible values:  
"Y" - register COM/DCOM server after download.  
"N" - do not register COM/DCOM server after download.  
For example: Register=Y  
If you omit this item, the default value “N” is used. |
| **progID** | Optional. Relevant only for COM/DCOM servers (EXE,OCX,DLL). The prog ID (the default string at the registry) for the COM/DCOM server.  
For example: ProgID=TestType.Class1 |
These parameters control the download of the control files to the client machine. This is done for every custom control file needed for this test type.

### Registering Custom Test Types Example

This example demonstrates how to register and use the custom controls and remote agent described in the previous examples in this chapter.

#### Registering Class IDs Inside MyTestType

The MyTestType example (see “TestType Example” on page 17) does not provide information about the class ID or the necessary remote agent ID. Once you have registered both the MyRunAgent and MyScriptViewer examples, revisit the MyTestType project and fix this problem.

First, you must get the registered class CLSIDs. From the **Start** menu, choose **Run**, and run **Regedit**. In the Registry Editor, choose **Edit > Find**, and search for **MyScriptViewer**. You find the class under the `HKEY_CLASSES_ROOT\CLSID\{XXXXXXXX-XXXX-...}` key, with the default string: `MyScriptViewer.UserControl`. The class ID string is the `{XXXXXXXX-XXXX-...}` part (with hexadecimal numbers instead of the Xs). It is the folder name in the left pane near the open folder icon.

Copy the class ID string into a text file for later use. Look at the Version sub-key and copy the version.

Repeat this procedure for **MyRunAgent** and **MyTestType**.

---

### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>version</strong></td>
<td>Optional. Relevant only for files that have compiled version information. If there is a COM server on the client machine with the same prog ID and the same version, the file is not downloaded. For example: <code>Version=1.48.9.2089</code></td>
</tr>
<tr>
<td><strong>checksize</strong></td>
<td>Optional. Relevant only if the ProgID and Version items are omitted. The file size in bytes. If there is a file with the same name and size in the destination directory, the file is not downloaded. For example: <code>CheckSize=4589</code></td>
</tr>
</tbody>
</table>
Chapter 2 • Implementing Testing Tool Integration

Next, open the MyTestType project with Visual Basic. Go to the Class_Initialize sub-routine, and type the MyScriptViewer and MyRunAgent class ID strings from above at the appropriate assignments of RemoteAgentCLSID and ScriptViewerCLSID variables.

Copy the Class ID string for MyScriptViewer to the "MyTestType" project, setting the variable "ScriptViewerCLSID". Replace "[Class ID for ActiveX script viewer]" with the class ID from the registry. For example:

ScriptViewerCLSID = "{CEC307CF-09A8-4E9B-B2D0-E4BBFAD1D93C}"

Copy the Class ID string for MyRunAgent to the "MyTestType" project, setting the variable "RemoteAgentCLSID". Replace "[Class ID for the Run Remote Agent]" with the class ID from the registry. For example:

RemoteAgentCLSID = "{19260D4B-EC7D-4868-9DCD-8B7FC64A370A}"

Do not use the ID strings from this example. Use the strings you copied from the registry on your development machine.

Recompile the MyTestType project by choosing File > Make MyTestType.exe from the menu bar. Save the project.

Note the new version number for MyTestType in the registry. Enter it in the text file where you are keeping the IDs and versions for later use.

Opening the war archive on the server

Go to the application directory under the Quality Center virtual directory on the server machine.

Create a temporary scratch directory in another location. Copy <QC Dir>\application\qcbin.war to the scratch directory and extract the files. This example assumes you created and extracted to D:\qctemp.
Registering MyTestType with test_type.ini

Edit test_types.ini file in the D:\qctemp\Install directory, and add the following lines:

[My-TEST]
CLSID={XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXX}

where you replace the {XXX......} part with the class ID string you found for MyTestType using Regedit. Save the file.

This file is downloaded to every registered client connecting to the server. You must delete this file from existing clients, so that it can be reloaded to these clients in its revised form.

Downloading the Files to the Client

Copy the three example files (MyTestType.exe, MyRunAgent.exe and MyscriptViewer.ocx) to the D:\qctemp\Install directory. Change their file extensions to xxx for the EXE files and .xco for the OCX file.

Open the D:\qctemp\setup_a_80mp.ini file. Note that this is a list of files to be downloaded to the client (including the test_type.ini file), numbered [File_1], [File_2], and so forth. Make a new section for each of the three files.

In this example, the last number before adding this example was 12. On your site the numbering may be different. Start with the next number after the highest file, [File_<current_max + 1>]. For “Version”, use the version numbers you saved above, when you recorded the class IDs from the registry on your development machine.

[File_13]
Register=Y
URLName=%URL%/Install/MyScriptViewer.xco
ShortName=MyScriptViewer.ocx
Description=My Script Viewer
ProgID=MyScriptViewer.UserControl1
Version=[the version is a sub key in the registry from within the class id]
[File_14]
Register=Y
URLName=%URL%/Install/MyTestType.xxx
ShortName=MyTestType.exe
Description=DOS Batch Test Type
ProgID=MyTestType.Class1
Version=[the version is a sub key in the registry from within the class id]

[File_15]
Register=Y
URLName=%URL%/Install/MyRunAgent.xxx
ShortName=MyRunAgent.exe
Description=My Remote Run agent
ProgID=MyRunagent.Class1
Version=[the version is a sub key in the registry from within the class id]

Save the file.

Recreating the war archive on the server
Update the D:\qctemp\qcbin.war file, so that it contains the three program files and the updated versions of the two ini files. Make a back-up copy of <QC Dir>\application\qcbin.war, then copy the updated archive, D:\qctemp\qcbin.war, into the application directory, overwriting the existing file.

Copy the <QC Dir>\application\qcbin.war archive to <QC Dir>\jboss\server\default\deploy\20qcbin.war, overwriting the existing 20qcbin.war. Note that the file is copied with a new name, 20qcbin.war.
Testing With MyTestType

The example custom test type is now ready for use.

---

**Note:** It is not recommended to use the station on which the custom test was compiled to run the client with the custom tests. If it is necessary to do so, delete all mentions of MyScriptViewer, MyTestType, and MyRunAgent from the system registry before running the client on the machine.

---

On the client machine, log off Quality Center. Delete the file `test_type.ini` from the client download directory, typically `C:\Program Files\Common Files\Mercury Interactive\TD2000_80`.

In this example, the batch file is created on the client before being transferred to the server. Ensure that the directory used by “CreateScriptTemplate” exists. In the sample above, the directory is `C:\temp`.

Open a browser and enter the URL, `http://<server:port>/qcbin`. Click the **Mercury Quality Center** link. You should see progress bars for each file you added to `setup_a_80mp.ini`.

Click the **Customize** button on the login page (in the upper left corner), and log in as the administrator. Click the **Customize Project Entities** link to open the Customize Project Entities dialog box. In the **Project Entities** list, expand the **Test** entity, expand the **System Fields** folder under it, and select **Type**. Click the **Goto List** button to open the Customize Project Lists dialog box. Click the **New Item** button, and type the name **My-TEST** to add it to the list of test types. Close the Project Lists dialog box, and click **OK** to close the Customize Project Entities dialog box. Log out of the Project Customization window. The **My-TEST** type is added to the test type options.

Log in to the Quality Center client. In the **Test Plan** module in the test plan tree, create a test of type **My-TEST**. Note that the icon to the left of the test name, in the test grid, indicating the test type, is a question mark (the default). Log out of the Quality Center client for the change to take effect.
Log in to the Quality Center client. In the **Test Plan** module in the test plan tree, select the test you added. Click the **Test Script** tab to display the test’s text inside the MyScriptViewer control you created earlier.

In the **Test Lab** module, select and add your test. Run the test. A minimized console window opens, the script commands are run, and the status field of the test is updated to Completed. Press **Enter** from the console window to close it.
Chapter 2 • Implementing Testing Tool Integration
Using the Quality Center API

The Quality Center application program interface (API) enables you to extend Quality Center functionality to your testing and reporting applications. Your applications can communicate with any Quality Center project and create, retrieve, and update project contents.

For details of the classes, methods, and properties exposed in the API, see the Quality Center Open Test Architecture API Reference.

This chapter describes:

➤ Integrating Your Applications with Quality Center
➤ Accessing Quality Center API Functions
➤ Downloading the OTAClient80.dll
➤ How the OTAClient80.dll Communicates with Quality Center
➤ Quality Center API Terminology

About the Quality Center API

Quality Center is a complete test management framework. At the center of this framework are projects that you create to store all the test requirements, test planning, test execution, and defect tracking information associated with each testing process. You access projects by using the Quality Center Login window, or by using external applications. These can include your configuration management, defect tracking, and custom testing tools.
You integrate external applications with Quality Center by using the Quality Center API. The Quality Center API objects expose COM-based interfaces that let you communicate directly with the Quality Center server to perform a variety of testing tasks. The Quality Center user interface uses this same API for all of its operations, such as connecting to a project, importing information from external applications to a project, and exporting information from a project to an external application.

The following chapters explain how to integrate external applications so that they can access and process information contained in projects.

**Integrating Your Applications with Quality Center**

Integrating Quality Center with other testing applications lets you extend Quality Center functionality to your execution, reporting, and productivity (word processing, spreadsheet, and presentation) applications.

For example, during the running tests stage you could use the Quality Center API to locate and retrieve tests created with third-party testing tools from the Quality Center test repository. The Quality Center API also lets you store all the data generated by the testing tool during each test run in the project.

After the running tests stage is complete, you can analyze this data with Quality Center reports and graphs, or export the results to an external application, such as a spreadsheet, to perform a detailed analysis. You can also use the Quality Center API to report any defects detected during a test to the project’s defect database automatically, and view this information in an external application.
Accessing Quality Center API Functions

You integrate external applications with Quality Center projects via COM objects. The definition of each object, including its properties, methods, and parameters, is packaged in the OTA (open test architecture) Client dynamic link library (OTAClient80.dll). After your application has performed an object function call, it is sent to the Quality Center server for processing. Note that you must download and register the OTAClient80.dll on each workstation that will communicate with the Quality Center server.
Chapter 3 • Using the Quality Center API

Downloading the OTAClient80.dll

The OTAClient80.dll is automatically downloaded to your workstation the first time you run Quality Center. Note that the OTAClient80.dll is not backward compatible with previous versions. An Open Test Architecture application must reference the OTAClient80.dll (also called the OTA COM 8.0 Type Library).

To download the OTAClient80.dll the first time you run Quality Center:

1 Open your Web browser and type the URL given to you by your system administrator.

2 Click the Quality Center link. The OTAClient 80.dll is automatically downloaded to the following folder:

<NT installation drive letter>\Program Files\Common Files\Mercury Interactive\TD2000_80

How the OTAClient80.dll Communicates with Quality Center

The OTAClient80.dll communicates with the Quality Center server either through Internet/intranet (HTTP) or through LAN (Microsoft’s DCOM). This enables your applications to receive information from, and send information to, your Quality Center projects.

Since Quality Center is a three-tier application, its server handles the communications between all applications that need to access the database, thus eliminating the need to install database clients on all the PCs that must access the database.
Quality Center API Terminology

Some of the terms used in the Quality Center project and the API differ from the terms used in the current Quality Center user interface.

<table>
<thead>
<tr>
<th>Quality Center Project</th>
<th>Quality Center API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defect</td>
<td>Bug</td>
</tr>
<tr>
<td>Attachment (cross reference)</td>
<td>Cros_ref</td>
</tr>
<tr>
<td>Test set</td>
<td>Cycle</td>
</tr>
<tr>
<td>Project</td>
<td>Database</td>
</tr>
<tr>
<td>Design steps</td>
<td>Dessteps</td>
</tr>
<tr>
<td>Test steps</td>
<td>Steps</td>
</tr>
<tr>
<td>Test Execution</td>
<td>Test Cycle</td>
</tr>
</tbody>
</table>
To fully utilize the Quality Center API to integrate external applications with Quality Center, you must understand the database design of the Quality Center project. The following sections explain the tables and fields that comprise the Quality Center project.

This chapter describes:

➤ Quality Center Projects
➤ Table Relationships
➤ Data Tables
➤ System Tables and Security Tables

Note: You should modify the database using only the Quality Center API. Do not modify it directly.

Quality Center Projects

This chapter describes the tables that comprise a Quality Center project and how these tables relate to one another. You can view each table and the fields it contains using the Site Administrator. For more information about viewing project tables using the Site Administrator, refer to the Mercury Quality Center Administrator’s Guide.

Each Quality Center project contains three types of tables: data tables, system tables, and security tables.
Data tables contain data entered by the users of the project at all stages of the testing process. For example, there are data tables that contain tests (Test table), test execution data (Run table), and defect information (Bug table). The Quality Center API interacts mainly with these tables, to which most of this chapter refers.

System tables contain information used internally by Quality Center. For example, there are system tables that contain information about fields in the project (System_Field table), the conditions under which defect reports are mailed to users (Mailcond table), and the host servers to which Quality Center is connected (Hosts table).

Security tables contain information used for generating, assigning, and controlling permissions within the project. For example, there are tables that control the actions that a user can perform (Actions table), the tables that a user can access and modify (Tables table), and the fields that the user can modify within the table (System_field table).

Table Relationships

The following diagram shows the relationship between the main data tables in a project, displaying their relevant fields. For clarity, the diagram is organized according to the Quality Center client entities.

Note: This is a partial description of the database and the relationships within it.
Chapter 4 • Quality Center Projects Data Structure
Data Tables

Data tables contain data entered by project users at all stages of the testing process. For example, the Test table lists each test and its name, ID, and type. Users can access this information through the main Quality Center window or through external applications that interface with the project, such as WinRunner. Quality Center projects include the following data tables:

<table>
<thead>
<tr>
<th>Table</th>
<th>Related Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Req</td>
<td>Requirements</td>
<td>Contains information about testing requirements.</td>
</tr>
<tr>
<td>Req_Cover</td>
<td>Requirements</td>
<td>Matches testing requirements to the tests and steps that cover them.</td>
</tr>
<tr>
<td>Test</td>
<td>Tests</td>
<td>Contains a list of all tests in Quality Center.</td>
</tr>
<tr>
<td>Dessteps</td>
<td>Design Steps</td>
<td>Contains information about design steps that are part of a planned test.</td>
</tr>
<tr>
<td>Cycle</td>
<td>Test Sets</td>
<td>Contains a list of all test sets in Quality Center.</td>
</tr>
<tr>
<td>Testcycl</td>
<td>Test Sets</td>
<td>Matches tests to test sets. Enables multiple instances of a single planned test within one test set.</td>
</tr>
<tr>
<td>Run</td>
<td>Test Execution</td>
<td>Contains information about test executions.</td>
</tr>
<tr>
<td>Step</td>
<td>Test Step Execution</td>
<td>Contains information about test step executions within a test execution.</td>
</tr>
<tr>
<td>Bug</td>
<td>Defects</td>
<td>Contains information about all the defects recorded in the project.</td>
</tr>
<tr>
<td>Bug_Tokens</td>
<td>Defects</td>
<td>Matches defect records to tokens used to search for similar defects.</td>
</tr>
<tr>
<td>Tokens</td>
<td>Defects</td>
<td>Contains tokens used to enable automated searching for similar defects.</td>
</tr>
<tr>
<td>Cros_Ref</td>
<td>Attachments</td>
<td>Files and URLs attached to any project entity.</td>
</tr>
</tbody>
</table>
You can customize a Quality Center project by adding user-defined fields. If you are using Oracle or Microsoft SQL Server databases, you can add up to 99 user-defined fields and 3 memo fields to each Quality Center entity.

<table>
<thead>
<tr>
<th>Table</th>
<th>Related Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>History</td>
<td>History</td>
<td>Contains record change information for the project entities.</td>
</tr>
<tr>
<td>All_Lists</td>
<td>Test Plan Tree</td>
<td>Contains the test plan tree with which the tests are associated. Also has a different role as a system table.</td>
</tr>
<tr>
<td>Step_Params</td>
<td>Test Parameters</td>
<td>Contains manual tests run parameters and their optional values.</td>
</tr>
<tr>
<td>Alerts</td>
<td>Traceability</td>
<td>Contains automatic generated alerts for data objects and user-defined follow-ups for data objects.</td>
</tr>
<tr>
<td>Cycl_Fold</td>
<td>Test Lab Folders</td>
<td>Contains the test lab tree with which the test sets are associated.</td>
</tr>
<tr>
<td>Rules</td>
<td>Customization/Traceability</td>
<td>Contains traceability identification rules which activate/deactivate rules and enable e-mail notifications.</td>
</tr>
</tbody>
</table>

Note: The maximum field size is 255 characters for Oracle or SQL Server databases.
The Req Table - Requirements

The Req table contains information about testing requirements. A project can have various requirements for which testing must be performed. Tests that cover elements of the requirement are assigned to the requirement in the Req_Cover table. For example, initialization can be a requirement, and all tests that involve initialization can be assigned to this requirement. Note that a test can be assigned to more than one requirement. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ_REQ_ID</td>
<td>The unique ID of the requirement.</td>
</tr>
<tr>
<td>RQ_FATHER_ID</td>
<td>The ID of the requirement for which the listed requirement is a sub-requirement.</td>
</tr>
<tr>
<td>RQ_ORDER_ID</td>
<td>The order in which the requirement appears among other requirements at the same level of the requirements hierarchy.</td>
</tr>
<tr>
<td>RQ_ISTEMPLATE</td>
<td>Indicates whether or not the requirement is a requirement template. Testers can design requirement templates as the basis for multiple requirements with common features.</td>
</tr>
<tr>
<td>RQ_REQ_COMMENT</td>
<td>Comments about the requirement.</td>
</tr>
<tr>
<td>RQ_REQ_REVIEWED</td>
<td>Indicates whether or not the User responsible for the project has reviewed the requirement.</td>
</tr>
<tr>
<td>RQ_REQ_PATH</td>
<td>The requirement path in the requirements hierarchy.</td>
</tr>
<tr>
<td>RQ_REQ_STATUS</td>
<td>The aggregated status of previous runs of tests in the requirement coverage. Indicates whether a test that covers the requirement fails. Possible values of this field: No Run, Failed, Passed, Not Completed, N/A. Additional user-defined values may be added to this field by the user.</td>
</tr>
<tr>
<td>RQ_REQ_PRIORITY</td>
<td>The requirement priority level. Possible values: 1-Low, 2-Medium, 3-High, 4-Very High, 5-Urgent.</td>
</tr>
<tr>
<td>RQ_REQ_TYPE</td>
<td>The requirement type (that is, system or functional).</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RQ_REQ_PRODUCT</td>
<td>The product for which the requirement is designed.</td>
</tr>
<tr>
<td>RQ_REQ_NAME</td>
<td>The name assigned to the requirement by the tester that designed the requirement.</td>
</tr>
<tr>
<td>RQ_REQ_AUTHOR</td>
<td>The user that designed the requirement.</td>
</tr>
<tr>
<td>RQ_REQ_VER_STAMP</td>
<td>The revision number of this requirement. Increases each time a change is made.</td>
</tr>
<tr>
<td>RQ_ATTACHMENT</td>
<td>Indicates whether the requirement has any attachments. The value of this field can be either Y or N.</td>
</tr>
<tr>
<td>RQ_REQ_DATE</td>
<td>The date the requirement was added to the project.</td>
</tr>
<tr>
<td>RQ_REQ_TIME</td>
<td>The time the requirement was added to the project.</td>
</tr>
<tr>
<td>RQ_NO_OF_SONS</td>
<td>The number of requirements that have this requirement as a father requirement (have their RQ_FATHER_ID field set to this requirement’s ID).</td>
</tr>
<tr>
<td>RQ_IS_FOLDER</td>
<td>Indicates if this requirement has folder-like behavior. Contains either Y or N.</td>
</tr>
<tr>
<td>RQ_VTS</td>
<td>The version time stamp. Indicates the time this record was last changed. The time stamp is according to the database server.</td>
</tr>
<tr>
<td>RQ_USER_01...102</td>
<td>You can add user-defined fields to customize the table (see page 63 for the maximum allowable number). You can also add up to 3 memo fields.</td>
</tr>
<tr>
<td>RQ_TASK_STATUS</td>
<td>For future use.</td>
</tr>
</tbody>
</table>
The Req_Cover Table - Requirements Coverage

The Req_Cover table matches requirements to the tests—and, optionally, test steps—that cover them. For each requirement, the table lists each test and test step that covers the requirement. For a diagram of these relationships, see “Table Relationships” on page 60.

The Req_Cover table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC_REQ_ID</td>
<td>The requirement ID. This is a reference to the Req table’s RQ_REQ_ID field.</td>
</tr>
<tr>
<td>RC_TEST_ID</td>
<td>The ID of a test that covers the requirement. This is a reference to the Test table’s TS_TEST_ID field.</td>
</tr>
<tr>
<td>RC_STEP_ID</td>
<td>The ID of a test step that covers the requirement. This is a reference to the Step table’s ST_STEP_ID field.</td>
</tr>
<tr>
<td>RC_ORDER_ID</td>
<td>The order in which this requirement coverage appears among other coverages of the requirement.</td>
</tr>
</tbody>
</table>

The Test Table - Tests

The Test table contains information on each test in the project, such as test ID, name, and type. When a new test is created, a new row is added to the Test table. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS_TEST_ID</td>
<td>The unique ID of the test.</td>
</tr>
<tr>
<td>TS_NAME</td>
<td>The name of the test assigned by the test developer.</td>
</tr>
<tr>
<td>TS_STEPS</td>
<td>The number of design steps in the test.</td>
</tr>
<tr>
<td>TS_PATH</td>
<td>The test directory path.</td>
</tr>
<tr>
<td>TS_SUBJECT</td>
<td>The ID of the subject folder in which the test resides.</td>
</tr>
<tr>
<td>TS_STATUS</td>
<td>The test’s current status. Possible values: Ready, Design, Imported, Repair. Additional user-defined values may be added to this field by the user.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TS_RESPONSIBLE</td>
<td>The user responsible for implementing the test.</td>
</tr>
<tr>
<td>TS_CREATION_DATE</td>
<td>The date on which the test was created.</td>
</tr>
<tr>
<td>TS_DESCRIPTION</td>
<td>A description of the test.</td>
</tr>
<tr>
<td>TS_TYPE</td>
<td>The type of test. For a description of possible test types, see <em>The Quality Center Open Test Architecture API Reference</em>.</td>
</tr>
<tr>
<td>TS_TIMEOUT</td>
<td>Reserved.</td>
</tr>
<tr>
<td>TS_ATTACHMENT</td>
<td>Indicates whether the test has any attachments. The value of this field can be either Y or N.</td>
</tr>
<tr>
<td>TS_USER_01...102</td>
<td>You can add user-defined fields to customize the table (see page 63 for the maximum allowable number). You can also add up to 3 memo fields.</td>
</tr>
<tr>
<td>TS_USER_HR_01...06</td>
<td>Not in use.</td>
</tr>
<tr>
<td>TS_ESTIMATE_DEVTIME</td>
<td>The time estimated to develop the test.</td>
</tr>
<tr>
<td>TS_TEST_VER_STAMP</td>
<td>The revision number of this record. Increases each time a change is made.</td>
</tr>
<tr>
<td>TS_EXEC_STATUS</td>
<td>Indicates the execution status of the test. Possible values: Not Completed, No Run, Passed, N/A, Failed.</td>
</tr>
<tr>
<td>TS_TEMPLATE</td>
<td>Indicates that this test is a test template. The value of this field can be either Y or N.</td>
</tr>
<tr>
<td>TS_STEP_PARAM</td>
<td>Not in use.</td>
</tr>
<tr>
<td>TS_VTS</td>
<td>The version time stamp. Indicates the time this record was last changed. The time stamp is according to the database server.</td>
</tr>
<tr>
<td>TS_VC_CUR_VER</td>
<td>The last checked-in version of the test. Used by the Version Control engine.</td>
</tr>
<tr>
<td>TS_TASK_STATUS</td>
<td>For future use.</td>
</tr>
</tbody>
</table>
The Dessteps Table - Design Steps

The Dessteps table contains information about design steps. Design steps are detailed, step-by-step instructions on how to execute a test, including the actions to perform on the application, the required input, and the expected output. When a new design step is created, a new row is added to the table. You can add new steps during the test run process and decide whether or not to save them to the Dessteps table. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS_TEST_ID</td>
<td>The ID of the test to which the step belongs. This is defined in the Test table TS_TEST_ID field.</td>
</tr>
<tr>
<td>DS_STEP_ID</td>
<td>The ID of the step within the test.</td>
</tr>
<tr>
<td>DS_STEP_ORDER</td>
<td>The order in which the step appears within the test.</td>
</tr>
<tr>
<td>DS_STEP_NAME</td>
<td>The name assigned to the step by the test developer.</td>
</tr>
<tr>
<td>DS_DESCRIPTION</td>
<td>The design step description.</td>
</tr>
<tr>
<td>DS_EXPECTED</td>
<td>The expected result string.</td>
</tr>
<tr>
<td>DS_USER_01...102</td>
<td>You can add user-defined fields to customize the table (see page 63 for the maximum allowable number). You can also add up to 3 memo fields.</td>
</tr>
<tr>
<td>DS_ATTACHMENT</td>
<td>Indicates whether the design step has any attachments. The value of this field can be either Y or N.</td>
</tr>
<tr>
<td>DS_LINK_TEST</td>
<td>The ID of a test to be run when this design step is executed. If no ID is present, no test will be run.</td>
</tr>
<tr>
<td>DS_HAS_PARAMS</td>
<td>Indicates whether this design step has parameters associated with it. This field can be either Y or N.</td>
</tr>
</tbody>
</table>
The Cycle Table - Test Sets

The Cycle table contains information about all the test sets in the Quality Center project. A test set is a group of tests designed to meet a specific testing goal. When a new test set is created, a new row is added to the Cycle table. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CY_CYCLE_ID</td>
<td>The ID of the test set.</td>
</tr>
<tr>
<td>CY_CYCLE</td>
<td>The test set name assigned by the creator of the test set.</td>
</tr>
<tr>
<td>CY_OPEN_DATE</td>
<td>The date the test set was opened.</td>
</tr>
<tr>
<td>CY_CLOSE_DATE</td>
<td>The date the test set was closed.</td>
</tr>
<tr>
<td>CY_STATUS</td>
<td>The current status of the test set. Possible values: Open, Closed.</td>
</tr>
<tr>
<td>CY_DESCRIPTION</td>
<td>The information contained in the Execution Flow tab.</td>
</tr>
<tr>
<td>CY_CYCLE_VER_STAMP</td>
<td>The revision number of this record. Increases each time a change is made.</td>
</tr>
<tr>
<td>CY_COMMENT</td>
<td>A description of the test set.</td>
</tr>
<tr>
<td>CY_ATTACHMENT</td>
<td>Indicates whether the test set has any attachments. The value of this field can be either Y or N.</td>
</tr>
<tr>
<td>CY_EXEC_EVENT_HANDLE</td>
<td>The actions to be executed in reaction to various execution events during the test set. This field is in a proprietary format.</td>
</tr>
<tr>
<td>CY_MAIL_SETTINGS</td>
<td>The mailing actions in reaction to various execution events during the test set. This field is in a proprietary format.</td>
</tr>
<tr>
<td>CY_USER_01...102</td>
<td>You can add user-defined fields to customize the table (see page 63 for the maximum allowable number). You can also add up to 3 memo fields.</td>
</tr>
</tbody>
</table>
The Testcycl Table - Tests in Test Sets

The Testcycl table contains information indicating which tests belong to which test sets. When a test is added to a test set, a new row is added to the table, indicating the Cycle_ID, Test_ID, and Test_Instance_ID. The table supports multiple instances of the same test in one test set through the TC_TEST_INSTANCE field. For a diagram of these relationships, see “Table Relationships” on page 60.

The Testcycl table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_TEST_ID</td>
<td>The ID of the test. This is defined in the Test table TS_TEST_ID field.</td>
</tr>
<tr>
<td>TC_CYCLE_ID</td>
<td>The ID of the test set in which the test resides. This is defined in the Cycle table CY_CYCLE_ID field. Note that the same test set ID can appear in many tests.</td>
</tr>
<tr>
<td>TC_TEST_INSTANCE</td>
<td>The number for this test instance inside the cycle. This field enables several test instances with the same TC_TEST_ID to reside within the same test set.</td>
</tr>
<tr>
<td>TC_CYCLE</td>
<td>Not in use.</td>
</tr>
<tr>
<td>TC_TEST_ORDER</td>
<td>The order in which the test appears within the test set.</td>
</tr>
<tr>
<td>TC_STATUS</td>
<td>The status of the last run of the test. Possible values: Not Completed, No Run, Passed, N/A, Failed. Additional user-defined values may be added to this field by the User.</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_TESTER_NAME</td>
<td>The user name of the person responsible for running the test.</td>
</tr>
<tr>
<td>TC_ACTUAL_TESTER</td>
<td>The name of the user that is actually executing the test.</td>
</tr>
<tr>
<td>TC_EXEC_DATE</td>
<td>The date on which the test was last executed.</td>
</tr>
<tr>
<td>TC_EXEC_TIME</td>
<td>The time at which the test was last executed.</td>
</tr>
<tr>
<td>TC_PLAN_SCHEDULING_DATE</td>
<td>The date on which the tester plans to next run the test.</td>
</tr>
<tr>
<td>TC_PLAN_SCHEDULING_TIME</td>
<td>The time at which the tester plans to next run the test.</td>
</tr>
<tr>
<td>TC_HOST_NAME</td>
<td>The name or IP address of the host server on which the test will be executed.</td>
</tr>
<tr>
<td>TC_EPARAMS</td>
<td>The testing tool configuration string, created by the testing tool itself.</td>
</tr>
<tr>
<td>TC_ATTACHMENT</td>
<td>Indicates whether the test instance has any attachments. The value of this field can be either Y or N.</td>
</tr>
<tr>
<td>TC_USER_01...102</td>
<td>You can add user-defined fields to customize the table (see page 63 for the maximum allowable number). You can also add up to 3 memo fields.</td>
</tr>
<tr>
<td>TC_TEST_VERSION</td>
<td>The current test version.</td>
</tr>
<tr>
<td>TC_TEST_CYCLE_VER_STAMP</td>
<td>The revision number of this record. Increases each time a change is made.</td>
</tr>
<tr>
<td>TC_EXEC_EVENT_HANDLE</td>
<td>The actions to be executed in reaction to various execution events during the test set run. This field is in proprietary format.</td>
</tr>
<tr>
<td>TC_VTS</td>
<td>The version time stamp. Indicates the time this record was last changed. The time stamp is according to the database server.</td>
</tr>
<tr>
<td>TC_TASK_STATUS</td>
<td>For future use.</td>
</tr>
</tbody>
</table>
Chapter 4 • Quality Center Projects Data Structure

The Run Table - Test Executions

The Run table contains information on test instance executions, such as the ID of the test run, the name of the test run, the test instance, and the time and date the run was performed.

When you run a test, a new row is created in the Run table and all test-specific design-steps are copied from the Dessteps table to the Steps table.

The Run table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RN_CYCLE_ID</td>
<td>The ID of the test set in which the test being run resides. This is defined in the Cycle table CY_CYCLE_ID field. Note that the same test set index can appear in many test runs.</td>
</tr>
<tr>
<td>RN_TEST_ID</td>
<td>The ID of the test being run. This is defined in the Test table TS_TEST_ID field. Note that a test may have more than one run.</td>
</tr>
<tr>
<td>RN_RUN_ID</td>
<td>The ID of the test run.</td>
</tr>
<tr>
<td>RN_RUN_NAME</td>
<td>The name assigned to the test run by the tester.</td>
</tr>
<tr>
<td>RN_HOST</td>
<td>The name or IP address of the host server on which the test was executed.</td>
</tr>
<tr>
<td>RN_STATUS</td>
<td>The current status of the test run. Possible values: Not Completed, No Run, Passed, N/A, Failed. Additional user-defined values may be added to this field by the User.</td>
</tr>
<tr>
<td>RN_EXECUTION_DATE</td>
<td>The date on which the test run was performed.</td>
</tr>
<tr>
<td>RN_EXECUTION_TIME</td>
<td>The time at which the test run was performed.</td>
</tr>
<tr>
<td>RN_DURATION</td>
<td>The duration of the test run.</td>
</tr>
<tr>
<td>RN_TESTER_NAME</td>
<td>The user name of the person who last executed the test.</td>
</tr>
<tr>
<td>RN_PATH</td>
<td>The directory path of the test run.</td>
</tr>
</tbody>
</table>
You can add user-defined fields to customize the table (see page 63 for the maximum allowable number). You can also add up to 3 memo fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RN_USER_01...102</td>
<td>You can add user-defined fields to customize the table.</td>
</tr>
<tr>
<td>RN_TEST_VERSION</td>
<td>The version of the application being tested.</td>
</tr>
<tr>
<td>RN_ATTACHMENT</td>
<td>Indicates whether the run has any attachments. The value of this field can be either Y or N.</td>
</tr>
<tr>
<td>RN_RUN_VER_STAMP</td>
<td>The revision number of this record. Increases each time a change is made.</td>
</tr>
<tr>
<td>RN_VTS</td>
<td>The version time stamp. Indicates the time at which this record was last changed. The time stamp is according to the database server.</td>
</tr>
<tr>
<td>RN_CYCLE</td>
<td>For future use.</td>
</tr>
<tr>
<td>RN_TEST_INSTANCE</td>
<td>The number of the test instance being run. This is defined in the Testycl table TC_TEST_INSTANCE field.</td>
</tr>
<tr>
<td>RN_OS_NAME</td>
<td>The name of the operating system on which the test run is running.</td>
</tr>
<tr>
<td>RN_OS_SP</td>
<td>The current service pack to which the operating system is updated.</td>
</tr>
<tr>
<td>RN_OS_BUILD</td>
<td>The current operating system build number.</td>
</tr>
<tr>
<td>RN_VC_LOKEDBY</td>
<td>The name of the User that locked the test. Used by the Version Control engine.</td>
</tr>
<tr>
<td>RN_VC_STATUS</td>
<td>The version control status of the test instance during execution. This field value can be one of the following: CHECKEDOUT, CHECKEDIN or GETTED.</td>
</tr>
<tr>
<td>RN_VC_VERSION</td>
<td>The test instance version during execution.</td>
</tr>
<tr>
<td>RN_DISK_STATUS</td>
<td>Not in use.</td>
</tr>
<tr>
<td>RN_MEM_STATUS</td>
<td>Not in use.</td>
</tr>
</tbody>
</table>
The Step Table - Test Steps

The Step table contains information on each test step performed during a test instance run. Test steps contain detailed, step-by-step instructions on how to execute a test. A step includes the actions to be performed on the application, required input, and actual output.

The Step table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_RUN_ID</td>
<td>The ID of the test run to which the test belongs. This is defined in the Run table RN_RUN_ID field. Note that the same Run ID can appear in many steps.</td>
</tr>
<tr>
<td>ST_STEP_ID</td>
<td>The ID of the test step.</td>
</tr>
<tr>
<td>ST_STEP_NAME</td>
<td>The name of the test step.</td>
</tr>
<tr>
<td>ST_STATUS</td>
<td>The status of the test step. Possible status values: Not Completed, No Run, Passed, N/A, Failed. Additional user-defined values may be added to this field by the User.</td>
</tr>
<tr>
<td>ST_EXECUTION_DATE</td>
<td>The date on which the test step was executed.</td>
</tr>
<tr>
<td>ST_EXECUTION_TIME</td>
<td>The time at which the test step was executed.</td>
</tr>
<tr>
<td>ST_DESCRIPTION</td>
<td>A description of the test step.</td>
</tr>
<tr>
<td>ST_EXPECTED</td>
<td>The expected result of the test step.</td>
</tr>
<tr>
<td>ST_ACTUAL</td>
<td>The actual result of the test step.</td>
</tr>
<tr>
<td>ST_PATH</td>
<td>The directory path of the test script containing the test step.</td>
</tr>
<tr>
<td>ST_LINE_NO</td>
<td>The line number on which the step appears in the test script.</td>
</tr>
<tr>
<td>ST_USER_01...102</td>
<td>You can add user-defined fields to customize the table (see page 63 for the maximum allowable number). You can also add up to 3 memo fields.</td>
</tr>
<tr>
<td>ST_STEP_ORDER</td>
<td>The order in which the step appears within the test.</td>
</tr>
<tr>
<td>ST_DESTEP_ID</td>
<td>Not in use. Use ST_DESIGN_ID instead.</td>
</tr>
</tbody>
</table>
The Bug Table - Defects

The Bug table contains information about each of the defects recorded in the Quality Center project, including defect ID, current status, and the developer responsible for correcting the defect.

The Bug table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG_BUG_ID</td>
<td>The ID of the defect record.</td>
</tr>
<tr>
<td>BG_CYCLE_ID</td>
<td>The index of the test set in which the defect was found. This is defined in the Cycle table CY_CYCLE_ID field.</td>
</tr>
<tr>
<td>BG_STATUS</td>
<td>The current status of the defect. The possible values of this field are: Open, Fixed, Closed, New, Rejected, Reopen. Additional user-defined values may be added to this field by the User.</td>
</tr>
<tr>
<td>BG_RESPONSIBLE</td>
<td>The name of the developer responsible for fixing the defect.</td>
</tr>
<tr>
<td>BG_PROJECT</td>
<td>The name of the project in which the defect was found.</td>
</tr>
<tr>
<td>BG_SUBJECT</td>
<td>The ID of the subject in the test plan tree to which the bug is related.</td>
</tr>
<tr>
<td>BG_SUMMARY</td>
<td>A summary of the defect.</td>
</tr>
<tr>
<td>BG_DESCRIPTION</td>
<td>A description of the defect.</td>
</tr>
</tbody>
</table>
## Chapter 4 • Quality Center Projects Data Structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG_DEV_COMMENTS</td>
<td>Comments about the defect by the developer responsible for the defect.</td>
</tr>
<tr>
<td>BG_REPRODUCIBLE</td>
<td>Indicates whether the tester was able to reproduce the defect.</td>
</tr>
<tr>
<td>BG_SEVERITY</td>
<td>The severity level of the defect. Possible values: 1-Low, 2-Medium, 3-High, 4-Very High, 5-Urgent.</td>
</tr>
<tr>
<td>BG_PRIORITY</td>
<td>The priority level of the defect. Possible values: 1-Low, 2-Medium, 3-High, 4-Very High, 5-Urgent.</td>
</tr>
<tr>
<td>BG_DETECTED_BY</td>
<td>The name of the tester who found the defect.</td>
</tr>
<tr>
<td>BG_TEST_REFERENCE</td>
<td>The ID of the test in which the defect was found. This is defined in the Test table TS_TEST_ID field.</td>
</tr>
<tr>
<td>BG_CYCLE_REFERENCE</td>
<td>The name of the test set in which the defect was found. This is defined in the Cycle table CY_CYCLE field.</td>
</tr>
<tr>
<td>BG_RUN_REFERENCE</td>
<td>The index of the test run in which the defect was found. This is defined in the Run table RN_RUN_ID field.</td>
</tr>
<tr>
<td>BG_STEP_REFERENCE</td>
<td>The ID of the test run step in which the defect was found. This is defined in the Step table ST_STEP_ID field.</td>
</tr>
<tr>
<td>BG_DETECTION_DATE</td>
<td>The date the defect was found.</td>
</tr>
<tr>
<td>BG_DETECTION_VERSION</td>
<td>The version in which the defect was detected.</td>
</tr>
<tr>
<td>BG_PLANNED_CLOSING_VER</td>
<td>The version in which the developer estimates the defect will be closed.</td>
</tr>
<tr>
<td>BG_ESTIMATED_FIX_TIME</td>
<td>The number of days the developer estimates will be required to fix the defect.</td>
</tr>
<tr>
<td>BG_ACTUAL_FIX_TIME</td>
<td>The number of days taken to fix the defect.</td>
</tr>
<tr>
<td>BG_CLOSING_DATE</td>
<td>The date the defect record was closed.</td>
</tr>
<tr>
<td>BG_CLOSING_VERSION</td>
<td>The version in which the defect record was closed.</td>
</tr>
</tbody>
</table>
### Chapter 4 • Quality Center Projects Data Structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG_TO_MAIL</td>
<td>Indicates whether a defect report should be mailed to users registered to receive such reports.</td>
</tr>
<tr>
<td>BG_ATTACHMENT</td>
<td>Indicates whether the defect record has any attachments. The value of this field can be either Y or N.</td>
</tr>
<tr>
<td>BG_USER_01...102</td>
<td>You can add user-defined fields to customize the table (see page 63 for the maximum allowable number). You can also add up to 3 memo fields.</td>
</tr>
<tr>
<td>BG_USER_HR_01...06</td>
<td>Not in use.</td>
</tr>
<tr>
<td>BG_BUG_VER_STAMP</td>
<td>A number indicating the revision number of this record. Increases each time a change is made.</td>
</tr>
<tr>
<td>BG_HAS_CHANGE</td>
<td>For backward compatibility.</td>
</tr>
<tr>
<td>BG_VTS</td>
<td>The version time stamp. Indicates the time this record was last changed. The time stamp is according to the database server.</td>
</tr>
<tr>
<td>BG_REQUEST_ID</td>
<td>Foreign key that links to the req table.</td>
</tr>
</tbody>
</table>
**The Bug_Tokens Table - Similar Defects**

The Bug_Tokens table matches defects stored in the Bugs table with tokens stored in the Tokens table. This enables Quality Center to search for similar defects. The following diagram shows the relationship between the Bug_Tokens table and the two tables related to it.

The Bug_Tokens table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT_BUG_ID</td>
<td>The ID of the defect.</td>
</tr>
<tr>
<td>BT_TOKEN_ID</td>
<td>The ID of the token.</td>
</tr>
</tbody>
</table>

**The Tokens Table - Tokens**

The Tokens table contains tokens that enable automated searching for similar defects. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TK_ID</td>
<td>The ID of the token.</td>
</tr>
<tr>
<td>TK_TOKEN</td>
<td>The name of the token.</td>
</tr>
</tbody>
</table>
The Cros_Ref Table - Attachments

The Cros_Ref table contains information about files and URLs attached to tests, test sets, and other project entities. The following diagram shows the relationship between the Cros_Ref table and other data tables. Note that this diagram excludes system tables.
The Cros_Ref table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR_REF_ID</td>
<td>The ID of the attachment within the Cros_Ref table.</td>
</tr>
<tr>
<td>CR_REFERENCE</td>
<td>The full path of the attached file within the file system or a URL. If it is a relative path, Quality Center searches it relative to the project directory that is defined for the project using the Site Administrator.</td>
</tr>
<tr>
<td>CR_OLE_IND</td>
<td>Not in use.</td>
</tr>
<tr>
<td>CR_REF_TYPE</td>
<td>The type of attachment. An attachment can be kept as a URL, as a file in the attachment repository, or as a link to a file.</td>
</tr>
<tr>
<td>CR_DESCRIPTION</td>
<td>A description of the attachment.</td>
</tr>
<tr>
<td>CR_VC_CUR_VER</td>
<td>For future use.</td>
</tr>
</tbody>
</table>
The History Table - Entity Changes History

The History Table shows changes made to records in Quality Center tables. When data changes, the History table is automatically updated. The following diagram shows how the History table records history information from the project.

Note: Not all changes made to a project entity are logged in the History table. Only changes made to specific, pre-defined fields are logged.
The History table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HS_TABLE_NAME</td>
<td>The name of the table for which the history record is logged.</td>
</tr>
<tr>
<td>HS_KEY</td>
<td>The string representation of the primary key of the record for which the history record is logged.</td>
</tr>
<tr>
<td>HS_COLUMN_NAME</td>
<td>The name of the field for which the history record is logged.</td>
</tr>
<tr>
<td>HS_CHANGE_DATE</td>
<td>The date on which the change being logged occurred.</td>
</tr>
<tr>
<td>HS_CHANGE_TIME</td>
<td>The time at which the change being logged occurred.</td>
</tr>
<tr>
<td>HS_CHANGER</td>
<td>The name of the user that made the change to the record.</td>
</tr>
<tr>
<td>HS_NEW_VALUE</td>
<td>The new value assigned to the field.</td>
</tr>
</tbody>
</table>
The All_Lists Table - Test Plan and List Values

The All_Lists table contains the test plan tree with the test TS_SUBJECT field, containing the AL_ITEM_ID of the node to which it belongs. It also functions as a system table, containing the list of all the values in all the drop down lists (combo boxes) displayed in the Quality Center grids. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL_ITEM_ID</td>
<td>The ID of the list item in the All_Lists table.</td>
</tr>
<tr>
<td>AL_FATHER_ID</td>
<td>The ID of the Parent folder of the item.</td>
</tr>
<tr>
<td>AL_DESCRIPTION</td>
<td>A description of the list item.</td>
</tr>
<tr>
<td>AL_NO_OF_SONS</td>
<td>The number of children of the item.</td>
</tr>
<tr>
<td>AL_SYSTEM</td>
<td>The type of list item. This can be one of the following:</td>
</tr>
<tr>
<td></td>
<td>R - read-only but can add sub-folders.</td>
</tr>
<tr>
<td></td>
<td>S - system field (Quality Center does not permit changes to this field).</td>
</tr>
<tr>
<td></td>
<td>A - all actions are available.</td>
</tr>
<tr>
<td></td>
<td>C - can change but not delete.</td>
</tr>
<tr>
<td>AL_ABSOLUTE_PATH</td>
<td>The list item path in the lists hierarchy.</td>
</tr>
<tr>
<td>AL_VIEW_ORDER</td>
<td>The order of the child item in the specified node.</td>
</tr>
<tr>
<td>AL_MEMO</td>
<td>A comment field.</td>
</tr>
<tr>
<td>AL_ATTACHMENT</td>
<td>Indicates whether the record has any attachments. The value of this field can be either Y or N.</td>
</tr>
<tr>
<td>AL_VTS</td>
<td>The version time stamp. Indicates the time this record was last changed. The time stamp is according to the database server.</td>
</tr>
</tbody>
</table>
The Step_Params Table - Manual Test Parameters

The Step_params table contains parameters for executing manual tests. The parameters can be assigned values, so that a manual test run configuration is completely documented. Note that a parameter can be assigned a value at several levels (namely: test, run, test set, or design step), and the contents of some fields can vary accordingly. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP_ENTITY</td>
<td>The type of the entity for which the parameter was entered. This field can contain one of the following: TEST, RUN, TESTCYCL, DESSTEP.</td>
</tr>
<tr>
<td>SP_KEY</td>
<td>The string representation of the primary key of the entity for which the parameter was entered.</td>
</tr>
<tr>
<td>SP_PARAM_NAME</td>
<td>This field contains the actual parameter name.</td>
</tr>
<tr>
<td>SP_PARAM_VALUE</td>
<td>The value that should be assigned to this parameter.</td>
</tr>
<tr>
<td>SP_ORIGIN_TEST</td>
<td>The ID of the test that was executed by the run. For Run entities only.</td>
</tr>
<tr>
<td>SP_VALUE_FROM_ENTITY</td>
<td>Not in use.</td>
</tr>
<tr>
<td>SP_VALUE_FROM_KEY</td>
<td>Not in use.</td>
</tr>
<tr>
<td>SP_REF_COUNT</td>
<td>The number of steps in which this parameter is used. For Test entities only.</td>
</tr>
<tr>
<td>SP_ASSIGNED_BY</td>
<td>Not in use.</td>
</tr>
<tr>
<td>SP_ASSIGN_DATE</td>
<td>Not in use.</td>
</tr>
<tr>
<td>SP_ASSIGN_TIME</td>
<td>Not in use.</td>
</tr>
</tbody>
</table>
The Alerts Table

The Alerts table contains information for managing automatic traceability alerts and user-defined follow-ups. Alerts are automatically generated using the data in the Rules table, while follow-ups are user-defined reminders that are set by the user.
The Alerts table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT_ID</td>
<td>The unique ID of the alert or follow-up.</td>
</tr>
<tr>
<td>AT_USER</td>
<td>The owner of the alert or follow-up.</td>
</tr>
<tr>
<td>AT_ENTITY_TYPE</td>
<td>The entity the alert was created for. Possible values are Bug, Test, or TestCycl.</td>
</tr>
<tr>
<td>AT_KEY1</td>
<td>The record on which the alert is based. The information in this field is the first primary key of the table referenced by the alert: Bug table - BG_BUG_ID, Test table - TS_TEST_ID, Testcycl table - TC_CYCLE_ID</td>
</tr>
<tr>
<td>AT_KEY2</td>
<td>The field that contains the primary key data of the Testcycl table: TC_TEST_ID. If the Bug table or Test table is selected, than this field is not used and the value is -1.</td>
</tr>
<tr>
<td>AT_KEY3</td>
<td>The string representation of the next primary key of the Testcycl table: TC_TEST_INSTANCE. If the Bug table or Test Table is selected, than this field is not used and the value is -1.</td>
</tr>
<tr>
<td>AT_DESCRIPTION</td>
<td>A description of the alert.</td>
</tr>
<tr>
<td>AT_DATE</td>
<td>The date the alert was sent by the system or the follow-up was configured to be sent.</td>
</tr>
<tr>
<td>AT_SENT_BY_EMAIL</td>
<td>Indicates that the follow-up was sent via e-mail. The value of this field can be either Y or N.</td>
</tr>
<tr>
<td>AT_ALERT_TYPE</td>
<td>The type of alert created. Possible values are: 1 - Follow-up, 2 - Automatic.</td>
</tr>
<tr>
<td>AT_ALERT_STATUS</td>
<td>The status of the alert sent. Possible values are: 0 - Unread, 1 - Read.</td>
</tr>
</tbody>
</table>
The Cycl_Fold Table
The Cycl_Fold table contains information regarding the test set tree, including the parent folder of each test set, and related tests. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF_ITEM_ID</td>
<td>The unique ID of the folder.</td>
</tr>
<tr>
<td>CF_ITEM_NAME</td>
<td>The name of the folder.</td>
</tr>
<tr>
<td>CF_ITEM_PATH</td>
<td>The encrypted path of the folder.</td>
</tr>
<tr>
<td>CF_FATHER_ID</td>
<td>The ID of the parent folder of the particular test set.</td>
</tr>
<tr>
<td>CF_VIEW_ORDER</td>
<td>For future use.</td>
</tr>
<tr>
<td>CF_WORKFLOW</td>
<td>For future use.</td>
</tr>
<tr>
<td>CF_ATTACHMENT</td>
<td>Indicates if the folder has an attachment. The value of this field can be either Y or N.</td>
</tr>
<tr>
<td>CF_VER_STAMP</td>
<td>The revision number of this record. Increases each time a change is made.</td>
</tr>
<tr>
<td>CF_VTS</td>
<td>The version time stamp. Indicates the time this record was last changed. The time stamp is according to the database server.</td>
</tr>
<tr>
<td>CF_DESCRIPTION</td>
<td>A description of the folder.</td>
</tr>
<tr>
<td>CF_NO_OF_SONS</td>
<td>The number of children of the parent folder.</td>
</tr>
</tbody>
</table>
The Rules Table

The Rules table contains information used together with the information in the Alerts table to generate automatic alert messages. Quality Center allows the user to activate/deactivate a rule and be notified by e-mail. An alert is generated when an activated rule becomes true. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RL_ID</td>
<td>The unique ID of the rule.</td>
</tr>
<tr>
<td>RL_CONDITION</td>
<td>For internal use.</td>
</tr>
<tr>
<td>RL_ACTION</td>
<td>For internal use.</td>
</tr>
<tr>
<td>RL_DESCRIPTION</td>
<td>A description of the rule.</td>
</tr>
<tr>
<td>RL_TO_MAIL</td>
<td>Indicates if the alert should be sent via e-mail. The value of this field can be either Y or N.</td>
</tr>
<tr>
<td>RL_IS_ACTIVE</td>
<td>Indicates if the rule is active. The value of this field can be either Y or N.</td>
</tr>
<tr>
<td>RL_IS_PREDEFINED</td>
<td>For future use.</td>
</tr>
</tbody>
</table>
System Tables and Security Tables

System tables contain information used internally by Quality Center. For example, the System_Field table contains information about project fields, the Mailcond table lists the conditions that govern whether defect reports are mailed to users, and the Groups table contains information defining user groups. Quality Center includes the following system tables:

<table>
<thead>
<tr>
<th>Table</th>
<th>Related Subject</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System_Field</td>
<td>Fields</td>
<td>Contains information about all project fields, including access permissions.</td>
</tr>
<tr>
<td>Hosts</td>
<td>Hosts</td>
<td>Contains information about host testing servers.</td>
</tr>
<tr>
<td>Host_Group</td>
<td>Hosts</td>
<td>Contains information about host groups.</td>
</tr>
<tr>
<td>Host_In_Group</td>
<td>Hosts</td>
<td>Matches host servers to host groups.</td>
</tr>
<tr>
<td>Mailcond</td>
<td>Mailing</td>
<td>Contains information for determining when defect reports are mailed to users.</td>
</tr>
<tr>
<td>Sequences</td>
<td>Sequences</td>
<td>Used to generate unique ID numbers.</td>
</tr>
<tr>
<td>Groups</td>
<td>User Groups</td>
<td>Contains information about user groups.</td>
</tr>
<tr>
<td>Users</td>
<td>Users</td>
<td>Contains information about users.</td>
</tr>
<tr>
<td>Dataconst</td>
<td>Project Constants</td>
<td>Contains the project constant values.</td>
</tr>
<tr>
<td>Locks</td>
<td>Fields Locks</td>
<td>Handles the locking of project entities.</td>
</tr>
<tr>
<td>Change</td>
<td>Change Integration</td>
<td>Contains changes made to fix defects. For backward compatibility. This table does not exist in new projects created with Quality Center 8.2 or later.</td>
</tr>
<tr>
<td>Change_Entry</td>
<td>Change Integration</td>
<td>Contains change details by file. For backward compatibility. This table does not exist in new projects created with Quality Center 8.2 or later.</td>
</tr>
</tbody>
</table>

Chapter 4 • Quality Center Projects Data Structure
Security tables contain information used for generating, assigning, and controlling access permission. For example, the Actions table defines the actions that users can perform, the Tables table defines the tables that users can access and modify, and the Transition Rules table defines the values that users can modify within tables. Quality Center includes the following security tables:

<table>
<thead>
<tr>
<th>Table</th>
<th>Related Subject</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change_Cover</td>
<td>Change Integration</td>
<td>Associates each change with the defect it fixed. For backward compatibility. This table does not exist in new projects created with Quality Center 8.2 or later.</td>
</tr>
<tr>
<td>Tran_Rules</td>
<td>Transition Rules</td>
<td>Contains information about the transition rules for the special fields of some entities.</td>
</tr>
<tr>
<td>Ver_Ctrl</td>
<td>Version Control</td>
<td>Version control locking management.</td>
</tr>
<tr>
<td>VC.xxx</td>
<td>Version Control</td>
<td>Shadow tables for version control. Represents as tables beginning with VC_.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table</th>
<th>Subject</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System_Field</td>
<td>Fields</td>
<td>Contains information about all project fields, including access permissions.</td>
</tr>
<tr>
<td>Actions</td>
<td>Actions</td>
<td>Lists all Quality Center actions and the types of access permissions associated with each action.</td>
</tr>
<tr>
<td>Tables</td>
<td>Tables</td>
<td>Lists Quality Center tables and the types of access permission associated with each table.</td>
</tr>
<tr>
<td>Modules</td>
<td>Module Access Management</td>
<td>Enables control over user group module access.</td>
</tr>
</tbody>
</table>

Security tables contain information used for generating, assigning, and controlling access permission. For example, the Actions table defines the actions that users can perform, the Tables table defines the tables that users can access and modify, and the Transition Rules table defines the values that users can modify within tables. Quality Center includes the following security tables:
The System_Field Table - Fields Properties

The System_Field table contains information about all project fields, such as field type and user customization label. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF_TABLE_NAME</td>
<td>The name of the table in which the field resides.</td>
</tr>
<tr>
<td>SF_COLUMN_NAME</td>
<td>The name of the column in which the field resides.</td>
</tr>
<tr>
<td>SF_COLUMN_TYPE</td>
<td>The type of column in which the field resides.</td>
</tr>
<tr>
<td>SF_USER_LABEL</td>
<td>A user-defined label for the field.</td>
</tr>
<tr>
<td>SF_EDIT_STYLE</td>
<td>The field edit style.</td>
</tr>
<tr>
<td>SF_EDIT_MASK</td>
<td>The field edit mask.</td>
</tr>
<tr>
<td>SF_IS_SYSTEM</td>
<td>Indicates whether the field is a system field.</td>
</tr>
<tr>
<td>SF_IS_CANFILTER</td>
<td>Indicates whether the field is filter-enabled.</td>
</tr>
<tr>
<td>SF_IS_KEY</td>
<td>Indicates whether the field is a database key field.</td>
</tr>
<tr>
<td>SF_KEY_ORDER</td>
<td>The field database key order.</td>
</tr>
<tr>
<td>SF_IS_EDIT</td>
<td>Indicates whether the field is editable.</td>
</tr>
<tr>
<td>SF_IS_ACTIVE</td>
<td>Indicates whether the field is active.</td>
</tr>
<tr>
<td>SF_IS_HISTORY</td>
<td>Indicates whether the history is kept of changes to the field.</td>
</tr>
<tr>
<td>SF_IS_MAIL</td>
<td>Indicates whether users on the notification list are notified when the field is changed.</td>
</tr>
<tr>
<td>SF_IS_VERIFY</td>
<td>Indicates whether the field requires verification.</td>
</tr>
<tr>
<td>SF_ROOT_ID</td>
<td>The All_Lists table AL_FATHER_ID that identifies the list of all possible values for the field.</td>
</tr>
<tr>
<td>SF_IS_BY_CODE</td>
<td>Indicates whether the field saves the Tree Node ID, rather than the passed value.</td>
</tr>
<tr>
<td>SF_IS_REQUIRED</td>
<td>Indicates whether the field is required.</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF_GRANT_MODIFY</td>
<td>Grants permission for the users in a user group to modify the field. This is a 32-bit mask that indicates whether the modify action is granted to a specific group. If bit X in this mask is assigned the number 1, the modify action is granted to all users in group X.</td>
</tr>
<tr>
<td>SF_IS_TRANSITION_LOGIC</td>
<td>Indicates whether transition logic exists for the field.</td>
</tr>
<tr>
<td>SF_USER_COLUMN_TYPE</td>
<td>The field user column type.</td>
</tr>
<tr>
<td>SF_IS_KEEP_VALUE</td>
<td>Indicates whether the last value of the field is stored.</td>
</tr>
<tr>
<td>SF_IS_CUSTOMIZABLE</td>
<td>Indicates whether the field is shown in the customization user interface.</td>
</tr>
<tr>
<td>SF_CAN_CHANGE_PERMISSIONS</td>
<td>Indicates whether the user can change the permission status of the field.</td>
</tr>
<tr>
<td>SF_OWNER_SENSIBLE</td>
<td>Indicates whether the field can be modified only by the owner of the entity.</td>
</tr>
<tr>
<td>SF_FIELD_SIZE</td>
<td>The field size.</td>
</tr>
<tr>
<td>SF_IS_VISIBLE_IN_NEW_BUG</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>SF_IS_VISIBLE_FOR_GROUPS</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>SF_IS_TO_SUM</td>
<td>Indicates whether there can be a summation of field contents for analysis. The value of this field can be either Y or N. Used for data fields only.</td>
</tr>
<tr>
<td>SF_IS_UNDER_VCS</td>
<td>Indicates whether this field is under version control. The value of this field can be either Y or N. Used for test fields only.</td>
</tr>
</tbody>
</table>
The Hosts Table - Testing Hosts

You can run automated tests on a local computer or on multiple remote hosts. A host is any computer connected to the network on which a testing tool has been installed. The Hosts table contains information on all hosts defined for the Quality Center project. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HO_NAME</td>
<td>The name of the host.</td>
</tr>
<tr>
<td>HO_DESCRIPTION</td>
<td>A description of the host.</td>
</tr>
<tr>
<td>HO_REX_SERVER</td>
<td>The name of the remote server installed on the host.</td>
</tr>
<tr>
<td>HO_ATTACHMENT</td>
<td>Indicates whether this host has any attachment associated with it. The value of this field can be either Y or N.</td>
</tr>
</tbody>
</table>

The Host_Group Table - Host Grouping

The Host_Group table lists all the host groups in the project. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GH_NAME</td>
<td>The name of the host group.</td>
</tr>
<tr>
<td>GH_ATTACHMENT</td>
<td>Indicates whether this host group has any attachment associated with it. The value of this field can be either Y or N.</td>
</tr>
<tr>
<td>GH_DESCRIPTION</td>
<td>The host group description.</td>
</tr>
</tbody>
</table>
The Host_In_Group Table - Host Grouping
The Host_In_Group table matches hosts to host groups. For each member in a group, it holds a record containing the group name and host name.

The Host_In_Group table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HG_HOST</td>
<td>The name of the host.</td>
</tr>
<tr>
<td>HG_GROUP</td>
<td>The name of the host group.</td>
</tr>
</tbody>
</table>

The Mailcond Table - Mailing Management
The Mailcond table contains information that determines when various users receive defect reports. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC_USER</td>
<td>The name of the user for which the condition exists. This is defined in the user combo table US_USERNAME field.</td>
</tr>
<tr>
<td>MC_CONDITION</td>
<td>The condition under the user receives defect reports by mail.</td>
</tr>
</tbody>
</table>

The Sequence Table - Generating IDs
The Sequence table is used to generate unique ID numbers, based on the highest ID for each table. The Sequence table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQ_SEQ_NAME</td>
<td>The sequence name. In most cases, this is the name of a data table (such as BUG or TEST).</td>
</tr>
<tr>
<td>SQ_SEQ_VALUE</td>
<td>The last sequence value.</td>
</tr>
</tbody>
</table>
The Groups Table - User Groups

The Groups table contains information about all of the user groups in the project. A user group consists of users that share a common set of access privileges. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_GROUP_ID</td>
<td>The ID of the group. The project administrator can create a maximum of 32 user groups and assign each of them an ID between 0 and 31.</td>
</tr>
<tr>
<td>GR_GROUP_NAME</td>
<td>The name of the user group assigned by the project administrator.</td>
</tr>
<tr>
<td>GR_IS_SYSTEM</td>
<td>Denotes if the field is one of five Quality Center default user groups. This field can be set to either Y or N.</td>
</tr>
<tr>
<td>GR_PREDEF_FILTER</td>
<td>Contains the pre-defined filter string for the group.</td>
</tr>
</tbody>
</table>

The Users Table - User Management

The Users table contains information about Quality Center users, such as user names and mailing addresses. Note that the table contains full details for only two generic users, Guest and Admin. For other generic users, only the US_USERNAME and US_GROUP fields are used.

The Users table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>US_USERNAME</td>
<td>The username assigned to the user by the project administrator.</td>
</tr>
<tr>
<td>US_MAIL_ADDRESS</td>
<td>The user's mailing address.</td>
</tr>
<tr>
<td>US_PASSWORD</td>
<td>The user's password.</td>
</tr>
<tr>
<td>US_GROUP</td>
<td>This is a 32-bit mask. Each digit in the mask represents membership in one of the user groups. One indicates that the user is a member of the corresponding user group. Zero indicates that the user is not a member of the user group. This allows the administrator to assign a user to a maximum of 31 user groups.</td>
</tr>
</tbody>
</table>
The Dataconst Table - Project Constants

The Dataconst table contains global values assigned to the project. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC_CONST_NAME</td>
<td>A string containing the constant name.</td>
</tr>
<tr>
<td>DC_VALUE</td>
<td>A string containing the constant value.</td>
</tr>
</tbody>
</table>

The global values set may be different from version to version and from installation to installation. Typical entries are:

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bug_file</td>
<td>Used by RDR.</td>
</tr>
<tr>
<td>checkouts_directory</td>
<td>The relative path of the version control checkout directory.</td>
</tr>
<tr>
<td>db_directory</td>
<td>The path of the project's related files. If db_directory is not empty, the value is used instead of the directory path defined for the project using the Site Administrator.</td>
</tr>
<tr>
<td>tests_directory</td>
<td>The relative path to test details and scripts.</td>
</tr>
<tr>
<td>unix_tests_directory</td>
<td>For future use.</td>
</tr>
<tr>
<td>VcsDb_directory</td>
<td>The relative path of the version control database.</td>
</tr>
<tr>
<td>version</td>
<td>The current Quality Center version number.</td>
</tr>
</tbody>
</table>
The Locks Table - Locking Entities

The Locks table handles locks on entities, preventing multiple users from modifying the same entity at the same time. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LK_OBJECT_TYPE</td>
<td>The entity type for which the lock is set.</td>
</tr>
<tr>
<td>LK_OBJECT_KEY</td>
<td>The string representation of the key of the specific entity that is locked.</td>
</tr>
<tr>
<td>LK_USER</td>
<td>The user locking the entity.</td>
</tr>
<tr>
<td>LK_SESSION_ID</td>
<td>The session ID in which the lock was imposed.</td>
</tr>
<tr>
<td>LK_CLIENT_MUID</td>
<td>Not in use.</td>
</tr>
<tr>
<td>LK_CLIENT_MACHINE_NAME</td>
<td>The name of the client machine that locked the entity.</td>
</tr>
<tr>
<td>LK_SERVER_MUID</td>
<td>Not in use.</td>
</tr>
<tr>
<td>LK_SERVER_MACHINE_NAME</td>
<td>The name of the server machine on which the lock was set.</td>
</tr>
<tr>
<td>LK_SERVER_IP_ADDRESS</td>
<td>The IP address of the server machine on which the lock was set.</td>
</tr>
<tr>
<td>LK_LOCK_TIME</td>
<td>The time at which the lock was set.</td>
</tr>
<tr>
<td>LK_SESSION_LAST_TOUCH</td>
<td>For future use.</td>
</tr>
</tbody>
</table>
The Tran_Rules Table - Transition Rules

The Tran_Rules table defines the transition rules for tables in the project. These rules determine the possible value transitions that users in a user group can perform in a specific field in a specific table. For example, the project administrator can specify that a group can change the status of a defect from Fixed to Closed. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TR_TABLE_NAME</td>
<td>The name of the table in which the field resides.</td>
</tr>
<tr>
<td>TR_FIELD_NAME</td>
<td>The name of the field for which the transition rule has been defined.</td>
</tr>
<tr>
<td>TR_GROUP_ID</td>
<td>The ID of the group to which the transition rule has been assigned. This is defined in the Group table GR_GROUP_ID field.</td>
</tr>
<tr>
<td>TR_RULES</td>
<td>The transition rule to be performed on the field.</td>
</tr>
</tbody>
</table>

The Ver_Ctrl Table - Version Control

The Ver_Ctrl table handles the version control locking information of the project. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VC_OBJECT_KEY</td>
<td>The string representation of the primary key of the locked object.</td>
</tr>
<tr>
<td>VC_USER_NAME</td>
<td>The username of the user locking the object.</td>
</tr>
<tr>
<td>VC_STATUS</td>
<td>The locked object status. This field is a string containing either CHECKOUT or GET.</td>
</tr>
<tr>
<td>VC_DATE</td>
<td>The date the object was locked.</td>
</tr>
<tr>
<td>VC_TIME</td>
<td>The time the object was locked.</td>
</tr>
<tr>
<td>VC_VERSION</td>
<td>The locked object’s version.</td>
</tr>
<tr>
<td>VC_COMMENTS</td>
<td>Comments regarding the locked object.</td>
</tr>
<tr>
<td>VC_OBJECT_TYPE</td>
<td>The entity type of the locked object.</td>
</tr>
</tbody>
</table>
The VC_ prefix Tables - Shadow Version Control Tables

Tables with names that begin with VC_ are the same as tables containing the same name without the VC_ prefix, except that tables with the VC_ prefix are checkout tables. These tables are used when an entity is checked out, or when the user is working on a specific version. The VC_ tables have one additional field, xxxVC_USER_NAME, where xxx indicates the appropriate table prefix (for example, TS_VC_USER_NAME in the VC_TEST table). These fields contain the name of the user that locked the entity.

One additional field is added to the VC_Test table only: TS_TASK_STATUS. This field contains the status of the task during testing, and is currently configured for future use.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VC_OBJECT_REFERENCE</td>
<td>The reference ID of the locked object. For a TEST this is the test ID.</td>
</tr>
<tr>
<td>VC_USER_1...102</td>
<td>You can add user-defined fields to customize the table (see page 63 for the maximum allowable number). You can also add up to 3 memo fields.</td>
</tr>
</tbody>
</table>
The Actions Table - Action Permissions

The Actions table contains information on different Quality Center actions a user can initiate in the project. The table contains the name of the action that can be performed and the user groups that are allowed to perform the action. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC_ACTION_NAME</td>
<td>The name of the action to be performed. The value for this field can be one of the following: ac_add_bug, ac_add_common_settings, ac_add_cover, ac_add_cycle, ac_add_cyclefolder, ac_add_cycle_to_cyclefolder, ac_add_desstep, ac_add_folder, ac_add_private_settings, ac_add_r&amp;d_comments, ac_add_req ac_add_reqfolder, ac_add_test, ac_add_test_to_testcycl, ac_change_password, ac_clear_history, ac_configure_mail, ac_copy_cycle, ac_copy_folder, ac_copy_cyclefolder, ac_create_rbr_files, ac_create_views, ac_create_wr_script, ac_customize_fields, ac_delete_bug, ac_delete_common_settings, ac_delete_cover, ac_delete_cycle, ac_delete_cycle_from_cyclefolder, ac_delete_cyclefolder, ac_delete_desstep, ac_delete_folder, ac_delete_private_settings, ac_delete_req, ac_delete_run, ac_delete_test, ac_delete_test_from_testcycl, ac_import_bugs_from_file, ac_import_bugs_from_mail, ac_import_bugs_settings, ac_import_wr_tests, ac_modify_bug, ac_modify_common_settings, ac_modify_cycle, ac_modify_desstep, ac_modify_folder, ac_modify_hosts, ac_modify_private_settings, ac_modify_req, ac_modify_reqfolder, ac_modify_run, ac_modify_test, ac_modify_test_in_testcycl, ac_move_folder, ac_move_cyclefolder, ac_remove_reqfolder, ac_reset_cycle, ac_run_auto_test, ac_run_manual_test, ac_send_all_qualified, ac_test_exec_params, ac_users, ac_wr_settings.</td>
</tr>
</tbody>
</table>
The Tables Table - Table Permissions

The Tables table contains the names of all the tables in the Quality Center project. It also contains information on the permissions associated with each table. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TB_TABLE_NAME</td>
<td>The name of the table</td>
</tr>
<tr>
<td>TB_GRANT_DELETE</td>
<td>Grants permission for the users in a user group to delete a record in the table. This is a 32-bit mask that indicates whether the action is granted to a specific group. If bit X in this mask is assigned the number 1, the delete action is granted to all Users in group X.</td>
</tr>
<tr>
<td>TB_GRANT_MODIFY</td>
<td>Grants permission for the users in a user group to modify a field in the table. This is a 32-bit mask that indicates whether the action is granted to a specific group. If bit X in this mask is assigned the number 1, the modify action is granted to all Users in group X.</td>
</tr>
</tbody>
</table>
Chapter 4 • Quality Center Projects Data Structure

The Modules Table - Module License Restrictions

Due to licensing constraints, the use of several modules is restricted to a limited number of Users. The Modules table handles group module access so that the number of Users will not violate these restrictions. The table contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TB_GRANT_ADD</td>
<td>Grants permission for the users in a user group to add a record to the table. This is a 32-bit mask that indicates whether the action is granted to a specific group. If bit X in this mask is assigned the number 1, the add action is granted to all Users in group X.</td>
</tr>
<tr>
<td>TB_OWNER_SENSIBLE_Delete</td>
<td>Specifies whether records in the table can only be deleted by their owners.</td>
</tr>
<tr>
<td>TB_OWNER_SENSIBLE_Modify</td>
<td>Specifies whether records in the table can only be modified by their owners.</td>
</tr>
<tr>
<td>TB_OWNER_FIELD_NAME</td>
<td>The column in the table representing the owner of the record.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_ID</td>
<td>The module ID number.</td>
</tr>
<tr>
<td>MD_NAME</td>
<td>The module name.</td>
</tr>
<tr>
<td>MD_GUID</td>
<td>The module unique GUID.</td>
</tr>
<tr>
<td>MD_DESC</td>
<td>The module description.</td>
</tr>
<tr>
<td>MD_VISIBLE</td>
<td>A bit mask, indicating which groups can access this module.</td>
</tr>
</tbody>
</table>
Index

A

action permissions
  action name 100
  Actions table 100
  adding records 102
  deleting records 102
  granting 101
  mask 101
  modifying fields 101
  modifying records 102
  owner 102
  table 101
  table name 101
  transition rules 98
  user groups 101
Actions table 100
ActiveX controls
  ExecConfiguration 13, 41
  ResultViewer 14, 39
  ScriptViewer 14, 31
Administrator’s Guide vii
Adobe Acrobat Reader vii
Alerts table 85
All_Lists table 83
API Reference
  Open Test Architecture viii
  Site Administrator Client viii
API, See Quality Center API
applications, integrating with Quality Center 54
attachments
  attachment ID 80
  Cros_Ref table 79
  description 80
  directory path 80
  host groups 93
  host servers 93
in defects 77
in design steps 68
in test runs 73
in test sets 69, 71
in test steps 75
requirements 65
tests 67
type 80

B

Books Online vii
Bug table 75
Bug_Tokens table 78

class ID 10
Columns dialog box 101
Contents and Index, Online Help viii
conventions. See typographical conventions
CreateScriptTemplate method
  TestType object 15
Cros_Ref (cross reference) table 79
custom controls 4
custom test types 10
  creating 11
  registering 46
custom testing tools. See testing tools
customer support online viii
customization 91
Cycl_Fold table 87
Cycle (test sets) table 69
Index

D

data analysis 54
data tables 62
   Alerts 85
   All_Lists 83
   Bug 75
   Bug_Tokens 78
   Cros_Ref (cross reference) 79
   Cycl_Fold 87
   Cycle (test sets) 69
   Dessteps 68
   History 81
   Req (requirements) 64
   Req_Cover (requirements coverage) 66
   Rules 88
   Run 72
   Step 74
   Step_Params 84
   Test 66
   Testcycl (test set per test) 70
   Tokens 78
database
   clients 56
   server. See database servers
   tables. See project database tables
database servers
   Microsoft SQL server 56
   Oracle 56
database_name parameter. See set_value
   function
Dataconst table 96
DCOM protocol 5, 19, 56
DCOM server 9
defects
   attachments 77
   Bug table 75
   Bug_Tokens table 78
   changes 77
   closed in version 76
   closing 76
   closing date 76
   date found 76
   description 75
   developer comments 76
   fix time, actual 76
   fix time, estimated 76
   ID 75, 78
   mailing 77
   mailing reports. See mailing change
   notification
   priority level 76
   project name 75
   reproducible 76
   request 77
   revision number of a record 77
   severity level 76
   status 75
   subject folder 75
   summary 75
   test found 76
   test run found 76
   test set found 75, 76
   test step found 76
   token ID 78
   user reporting 76
   user responsible for 75
   user-defined 77
   version found 76
   version time stamp 77
design steps
   attachments 68
   description 68
   Dessteps table 68
   expected result 68
   ID 68
   name 68
   number of 66
   order 68
   parameters 68
   step ID 68
   test ID 68
   user-defined 68
Dessteps (design steps) table 68
documentation set vii
documentation updates ix
Index

E
error handling
returning most recent error message 13
examples
registering custom test types 46
RemoteAgent 24
ScriptViewer 33
TestType COM class 17
ExecConfiguration ActiveX control 41–43
Init method 42
ShowExecConfiguration method 43
ShowExecConfigurationEX method 42
ExecConfiguration class 13

F
fields
active status 91
changing permission status 92
column name 91
column type 91
customizable or not 92
customization information 91
database key 91
database key order 91
default mask 91
default style 91
filter-enabled 91
history records 91
labels 91
linked values 91
mailing change notification. See
mailing change notification
modification privileges 92
owner 92
properties 91
required or not 91
saving tree node ID 91
size 92
storing last value 92
sum 92
system 91
table 91
transition logic 92
type 91
user column type 92
user-defined 65
verification required 91
version control 92

G
get_status method
RemoteAgent DCOM server 22
graphs 2, 54
Groups table 95

H
history
management 81
history records
change date 82
change time 82
column name 82
index 82
new value 82
storing 91
table 81
table name 82
user 82
History table 81
host groups
attachments 93
description 93
Host_Group table 93
Host_In_Group table 94
list of 93
member hosts 94
name 93, 94
host servers 56
attachments 93
description 93
Hosts table 93
management 93
name 93
remote execution server 93
test execution host 71
Index

Host_Group table 93
Host_In_Group table 94
Hosts table 93

I
IDispatch interface 19
Init method
   ExecConfiguration ActiveX control 42
   ResultViewer ActiveX control 39
   ScriptViewer ActiveX control 31
   TestType object 14
Installation Guide vii
is_host_ready method
   RemoteAgent DCOM server 19

L
lists
   Alerts table 85
   All_Lists table 83
   attachments 83
   child 83
   comments 83
   Cycl_Fold table 87
   description 83
   father ID 83
   item information 83
   list type 83
   path 83
   Rules table 88
   Step_Params table 84
   view order 83
locks
   client machine 97
   entity key 97
   entity type 97
   IP address 97
   server machine 97
   session ID 97
   time 97
   user 97
Locks table 97

M
mail conditions. See Mailcond table
Mailcond table 94
mailing change notification 77, 91, 94
Mercury Interactive on the Web viii
Mercury Quality Center
documentation set vii
Mercury Quality Center Administrator’s Guide vii
Mercury Quality Center Business Process Testing User’s Guide vii
Mercury Quality Center Installation Guide vii
Mercury Quality Center Tutorial vii
Microsoft SQL 56
modules
description 102
GUID 102
ID number 102
module access 102
name 102
Modules table 102

O
online help viii
online resources vii
online support viii
Open Test Architecture API Reference viii
Oracle 56
OTAClient.dll 55
   communicating with the Quality Center server 56
downloading 56
installing 55

P
parameter
   entity 84
   name 84
   number of steps 84
   primary key 84
   test ID 84
   value 84
password parameter. See set_value function
plan_status parameter. See set_value function
pre-defined test types 10
project database tables 59–102
  Actions 100
  Alerts 85
  All_Lists 83
  Bug (defects) 75
  Bug_Tokens 78
  Cros_Ref (cross reference) 79
  Cycl_Fold 87
  Cycle (test sets) 69
data 62
  Dataconst 96
  Dessteps (design steps) 68
  Groups 95
  History 81
  Host_Group 93
  Host_In_Group table 94
  Hosts 93
  Locks 97
  Mailcond (mail conditions) 94
  Modules 102
  Req (requirements) 64
  Req_Cover (requirements coverage) 66
  Rules 88
  Run 72
security 60, 90
  Sequence 94
  Step 74
  Step_Params 84
  system 89
  System Field 91
table relationships diagram 60
  Tables 101
  Test 66
  TestCycl (tests assigned to test sets) 70
  Tokens 78
  Tran_Rules (transition rules) 98
  Users 95
  VC_...(shadow version control) 99
  Ver_Ctrl (version control) 98
project_name parameter. See set_value function
projects
  accessing database 55
data tables 62
exporting information 54
importing information 54
security tables 60, 90
system tables 89
table relationships 60
test sets 69
Q
Quality Center
documentation set vii
Quality Center API
  accessing Quality Center API functions 55
  integrating applications with Quality Center 4, 54
  overview 53–58
terminology 58
Quality Center client 4
Quality Center project terminology 58
Quality Center server 56
Quality Center server, See also server connection 4
R
Readme vii
registering custom test types 46
remote agent 4
class ID 13
  creating 19–30
  example 24
  IDispatch interface 19
remote execution server 93
remote hosts 10
remote test execution 2, 10
RemoteAgent DCOM server 19–30
  get_status method 22
  is_host_ready method 19
  run method 22
  set_value method 20
  stop method 22
reports 2, 54
Req (requirements) table 64
Req_Cover (requirements coverage) table 66
Index

requirements
attachment 65
author 65
commment 64
date 65
directory path 64
father 64
folder 65
ID 64, 66
name 65
order 64
priority 64
product 65
Req table 64, 66
Req_Cover table 66
requirement coverage 66
review status 64
revision number 65
sons 65
status 64
steps 66
task status 65
templates 64
test coverage 66
time 65
type 64
version time stamp 65
responsible parameter. See set_value function
ResultViewer ActiveX control 39–40
class ID 14
displaying test results 39, 40
Init method 39
ShowResult method 40
ShowResultEx method 39
RQ 65
Rules table 88
run method
RemoteAgent DCOM server 22
Run table 72
runner_result_dir parameter. See set_value function
scheduler_version parameter. See set_value function
ScriptViewer ActiveX control 31–38
class ID 14
displaying test scripts 32
example 33
Init method 31
SaveScript method 32
saving test scripts 32
ShowTest method 32
search
similar defects 78
security tables 60, 90
Actions 100
Modules 102
System Field 91
Tables 101
sequence
name 94
value 94
sequence generation diagram 94
Sequence table 94
server connection 56
sessions
initiating 12, 41
terminating 41
set_value function 41
set_value method
RemoteAgent DCOM server 20
ShowExecConfiguration method
ExecConfiguration ActiveX control 43
ShowExecConfigurationEX method
ExecConfiguration ActiveX control 42
ShowResult method
ResultViewer ActiveX control 40
ShowResultEx method
ResultViewer ActiveX control 39
ShowTest method
ScriptViewer ActiveX control 32
Site Administrator Client API Reference viii
Site Administrator, viewing project tables 59
Step table 74
Step_Params table 84

S
SaveScript method
ScriptViewer ActiveX control 32
stop method
- RemoteAgent DCOM server 22
storing tests 54
subject parameter. See set_value function
support online viii
sys_computer_name parameter. See set_value function
sys_user_name parameter. See set_value function
System Field table 91
system tables 89
- Dataconst 96
  - Groups 95
  - Host_Group 93
  - Host_In_Group 94
  - Hosts 93
  - Locks 97
  - Mailcond (mail conditions) 94
  - Sequence 94
  - System Field 91
  - Tran_Rules (transition rules) 98
  - Users 95
  - VC...(shadow version control) 99
  - Ver_Ctrl (version control) 98

T
table relationships 60
tables
  - data 62
  - security 90
  - system 89
Tables table 101
task status
  - requirements 65
test sets 70
tests 67
TDAPI_host_name parameter. See set_value function
TDConnection object 12
technical support online viii
templates, test requirements 64
Test Lab module 1
test run scheduler 10
test runs
  - attachments 73
  - current status 72
directory path 72
duration 72
execution date 72
execution time 72
host name 71
host server 72
ID 72
locked tests (version control) 73
name 72
operating system 73
operating system build number 73
revision number of record 73
Run table 72
service pack 73
status of last run 70
test ID 72
test instance during execution (version control) 73
test instances 73
test instances (version control) 73
test set index 72
test version 73
user responsible for 72
user-defined 73
version time stamp 73

test sets 1
  - actual tester 71
  - attachments 69, 71
close date 69
comments 69
configuration 71
current version of test 71
Cycle table 69
date of next test run 71
description 69
events 69
execution events 71
folder ID 70
ID 69
mail 69
name 69
open date 69
order of tests 70
revision number 69
revision number of a record 71
Index

status 69
task status 70
test ID 70
test instances 70
test set ID 70
TestCycl table 70
tester name 71
time of next test run 71
user-defined 69, 71
version time stamp 70, 71
test steps
actual results 74
attachments 75
description 74
design step ID 74, 75
directory path 74
execution date 74
execution time 74
expected results 74
ID 74
line number 74
name 74
order 74
status 74
Step table 74
test ID 75
test run ID 74
Test table 66
test types
creating 11
custom 10, 11
ExecConfiguration ActiveX control 13, 41
overview 10
pre-defined 10
registering 43
ResultViewer ActiveX control 14, 39
ScriptViewer ActiveX control 14, 31
table entry 67
test_id parameter. See set_value function
test_instance parameter. See set_value function
test_name parameter. See set_value function
test_path parameter. See set_value function
test_set_id parameter. See set_value function
test_set_user1-6 parameter. See set_value function
test_type.ini file 7, 43
test_user1-6 parameter. See set_value function
Testcycl table 70
testcycle_id parameter. See set_value function
TestDirector, see Quality Center
tester_name parameter. See set_value function
testing tools
communicating with Quality Center 3
configuration settings 41
integrating with Quality Center 1–8, 9–51
retrieving information from the Quality Center database 20
running tests 22
tests
analysis 2, 54
attachments 67
configuration 41
creation date 67
description 67
design steps 66
directory path 66
displaying results 39, 40
displaying scripts 32
estimated development time 67
execution date 71
execution status 67
execution time 71
ID 66
name 66
repository 54
responsible user 67
results 2
revision number 67
running 22
saving scripts 32
script template 15
status 66
stopping 22
storing 54
subject folder 66
task status 67
template 67
Test table 66
user-defined 67
version control 67
version time stamp 67
TestType COM class 8
example 17
TestType object 12
TestType mechanism 6
TestType object 12
  CreateScriptTemplate method 15
  Init method 14
third-party testing tools. See testing tools
Tokens table 78
Tran_rules (transition rules) table 98
transition logic 92, 98
tsettest_name parameter. See set_value function
Tutorial vii
typographical conventions in this guide ix

U
updates, documentation ix
user groups
  action permissions 101
  field name 98
  filter 95
  ID 95, 98
  name 95
  system 95
  table 95, 98
  transition rule 98
user_name parameter. See set_value function
users
  address 96
  contact information 95
  customization 91
  management 95
  name 95, 96
  password 95

phone 96
user groups 95
Users table 95

V
VC_... table 99
Ver_Ctrl table 98
version control
  comments 98
  date 98
  entity type 98
  object’s version 98
  primary key 98
  reference ID 99
  status 98
  table 98
  time 98
  user name 98
  user-defined 99

W
What’s New vii
WinRunner
class ID 10
ini file(wrun.ini) 10
Index