Borland®

Silk Performer 16.5

Citrix Tutorial

Borland Software Corporation
700 King Farm Blvd, Suite 400
Rockville, MD 20850

2015-09-10

Contents

Citrix XenApp Support

Silk Performer provides record and replay support for the testing of applications that are hosted via Citrix XenApp session and application virtualization technologies.

Citrix facilitates real-time access to shared applications over networks and the Internet. Remote access to Citrix-enabled applications can be over DSL, T1, ISDN, or dial-up. Citrix enables multiple users to run shared applications simultaneously. Communication between Citrix plug-ins (clients) and servers consists of exchange of user inputs (keyboard/mouse) and screen shots.

Silk Performer supports both Citrix server recording and Citrix hosted application recording. Citrix plug-ins offer access to specific applications and multiple desktop windows.

The Citrix Web Interface enables users to access hosted XenApp applications and XenDesktop virtual desktops without connecting directly to host computers. Users access these resources either via a Web browser or the Citrix online plug-in.

Silk Performer offers two Citrix project types:

- *Citrix* - Used to test hosted applications that are accessed via Citrix plug-ins (clients)
- *Citrix Web Interface* - Used to test Citrix-enabled applications that are published via Citrix Web interface

The PDF-based Citrix XenApp Tutorial walks you through the entire process of testing Citrix-enabled applications. The tutorial is available in Silk Performer at **Start** > **All Programs** > **Silk** > **Silk Performer 16.5** > **Documentation** > **Tutorials** > **Citrix XenApp** .

Note: Silk Performer Help includes all of the content that is included in the Citrix XenApp Tutorial.

Citrix Program Neighborhood

Rather than relying on the Web Interface to distribute applications, earlier versions of Citrix use the Windows-based Citrix Program Neighborhood. The Citrix Program Neighborhood allows users to view a listing of all available applications that are by Citrix XenApp on-demand application delivery servers. An enumeration of available resources takes place automatically each time you launch the Citrix Program Neighborhood via the Citrix Browser Service.

Important: When applications are distributed via the Citrix Program Neighborhood you must configure the Citrix published application recording service to start on your system or the Citrix Program Neighborhood will not be able to update your applications list.

Defining Projects

The first step in conducting a Citrix load test is to define the basic settings for your Silk Performer project. Following that you need to perform prerequisite XenApp server configurations.

Configuring the Citrix client software

Before defining a Citrix project you must install the Citrix client software (Citrix Receiver) and configure how the Citrix server is to handle time-outs. To download the Citrix client software, navigate to *http://www.citrix.com*.

Configuring the Citrix server depends on the version you are using. Following is the procedure for configuring Citrix Server 4.0.

Note: To configure Citrix Server version 4.5 or higher, go to **Start** > **Programs** > **Administrative Tools** > **Terminal Services Configuration** and double-click the **ica-tcp** option.

1. Go to **Start** > **Programs** > **Citrix** > **Administration Tools** and launch the **Citrix Connection Configuration Tool**.
2. Double-click **ica-tcp** connection.
3. On the following dialog box, click **Advanced**. The **Advanced Connection Settings** dialog box appears.
4. Set the **Disconnection timeout** limit. 1 minute is the recommended timeout.
5. Set the **Idle timeout** limit. 5 minutes is the recommended timeout.
6. Select **reset** from the **On a broken or timed-out connection** drop list.

   If this option is not selected, sessions will remain open after replay stops due to broken or timed-out connections. This will generate replay problems when users next log into sessions as scripts will continue from the point where they left off in the previous session. In such instances sessions need to be ended manually.

Defining Your Citrix XenApp Project

1. Click **Start here** on the Silk Performer workflow bar.

   Note: If another project is already open, choose **File** > **New Project** from the menu bar and confirm that you want to close your currently open project.

   The **Workflow - Outline Project** dialog box opens.
2. In the **Name** text box, enter a name for your project.
3. Enter an optional project description in **Description**.
4. From the **Type** menu tree, select **Terminal Services** > **Citrix** or **Terminal Services** > **Citrix Web Interface**.

   The **Citrix** application type uses the Silk Performer Citrix Recorder to test the delivery of Citrix-enabled applications that are accessed via any Citrix online or offline plug-in.

   The **Citrix Web Interface** application type is used to test the delivery of Citrix-enabled applications that are accessed via the Citrix Web Interface.
5. Click **Next** to create a project based on your settings.

The **Workflow - Model Script** dialog box appears.

Creating a Citrix Plug-In Test Script

The easiest approach to recording user actions via Citrix XenApp plug-ins (clients) that connect directly to Citrix XenApp servers and then creating a test script is to use the Silk Performer Recorder, the Silk Performer engine used for capturing and recording traffic and generating test scripts.

The Silk Performer Recorder captures and records the traffic between Citrix XenApp plug-ins and the server under test. When recording is complete, the Silk Performer Recorder automatically generates a test script based on the recorded traffic. Scripts are written in the Silk Performer scripting language, Benchmark Description Language (BDL).

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. In the **Application Profile** list, select `Silk Performer Citrix Recorder` to record a Citrix XenApp-enabled application via the Silk Performer Citrix Recorder application.

   This application profile is appropriate for testing Citrix plug-ins that connect directly to Citrix XenApp servers.
3. *Optional:* If you have an ICA file that defines your server connection parameters, type the full path of the ICA file into the **Command line** field.

4. Click **Start recording**.
5. The Silk Performer Recorder then opens in minimized form along with the Silk Performer Citrix Recorder.

   If correct login credentials are not available at startup, the **Connect** dialog will open. The **Connect** dialog is also accessible via the **Connect** button in the upper left corner of the Silk Performer Citrix Recorder.

   Note: If you have an ICA file that defines your server connection parameters, select **ICA File** and browse to the ICA file to skip past all of the **Login** fields.
6. On the **Connect** dialog in the **Server** field, enter the name of the Citrix XenApp server that is to be recorded.
7. Complete the **User name**, **Password**, and **Domain** fields for the server under test.
8. Type the name of the hosted application in the **Application** field.
9. The **Client name** field enables you to specify a client name for your session. The default client name for the Citrix recorder is `SP_Recorder` (the default client name for the Citrix Player is `SP_User_x`). If you specify a different client name for recording, then a `CitrixSetClientName` function will be inserted into the script. In such cases you must customize the client name value, otherwise all users will use the same client name, which may lead to replay problems.
10. Select the desired **Color depth** for recording.
11. Select the desired screen **Resolution** for recording.

    Note: Do not change the resolution after recording your script, as replaying with a different resolution setting may cause mouse actions to fail on repositioned elements.
12. Click **Connect** to begin the Citrix XenApp session.
13. Interact with the shared desktop in the Citrix Recorder in the same way that you want your virtual users to act during the test. For example, you can click links, open applications, and enter data. Your actions will be captured and recorded by the Citrix Recorder.

    The Citrix Recorder supports session sharing, allowing you to start additional published applications in the existing session (by clicking **Run Application** on the Citrix Recorder toolbar). Click **Select Window** to switch between applications.

    The following example Citrix XenApp session includes simple Excel calculations in which the mouse and keyboard are used to open Excel, enter new data values, insert an AutoSum formula, select a screen region, edit the AutoSum formula, and close Excel.
14. When you are done recording your Citrix XenApp session, click **Stop** on the Silk Performer Citrix Recorder. The **Generating script** progress window opens followed by the **Save As** dialog.
15. Save the script with a meaningful name.
16. A newly generated test script that is based on your recorded actions appears in the Silk Performer script editor window.

Citrix Web Interface Sessions (NFuse)

Citrix Web Interface software (previously known as NFuse) provides Web access to Java, UNIX, and Windows applications that are hosted via Citrix application server software. While Citrix offers server-side control of hosted applications, Citrix Web Interface makes applications accessible through a Web browser interface (Internet Explorer, version 5.5. or higher).

For technical support and questions regarding Citrix Web Interface, go to *http://support.citrix.com*

Defining a Citrix Web Interface Project

1. Click **Start here** on the Silk Performer workflow bar.

   Note: If another project is already open, choose **File** > **New Project** from the menu bar and confirm that you want to close your currently open project.

The **Workflow - Outline Project** dialog box opens.

2. In the **Name** text box, enter a name for your project.
3. Enter an optional project description in **Description**.
4. In the **Type** menu tree, select **Terminal Services** > **Citrix Web Interface**.
5. Click **Next** to create a project based on your settings.

The **Workflow - Model Script** dialog box appears.

Creating a Citrix Web Interface Test Script

The easiest approach to creating a test script for a Citrix Web Interface session is to use the Silk Performer Recorder, the Silk Performer engine for capturing and recording traffic and generating test scripts.

The Silk Performer Recorder captures and records traffic between a Citrix Web Interface client application (Internet Explorer, version 5.5 or higher) and the server under test. When recording is complete, the Silk Performer Recorder automatically generates a test script based on the recorded traffic. Scripts are written in the Silk Performer scripting language, Benchmark Description Language (BDL).

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. In the **Application Profile** list, select `Citrix Web Interface`.

   The `Citrix Web Interface` application profile is only appropriate for testing Citrix Web Interface/ NFuse sessions.
3. Click **Start recording**.
4. The Silk Performer Recorder then opens in minimized form along with Internet Explorer. Enter the name of the Citrix server in the Internet Explorer **Address** field and click **Enter**. To see a report of the actions that occur during recording, maximize the Silk Performer Recorder dialog by clicking **Change GUI size** on the Recorder toolbar.
5. To log into the Citrix Web Interface, enter your **Username**, **Password**, and **Domain** into the Citrix Web Interface login screen. Contact your system administrator if you do not have this information.
6. Click **Log In**.
7. The application portal appears. This portal contains the applications that have been published for shared use. Select the hosted application you want to record.
8. The hosted application appears in the Silk Performer Citrix Recorder. Interact with the shared application in the Citrix Recorder in the same way that you want your virtual users to behave during the test. Your actions will be captured by the Citrix Recorder and generated into a BDL script.
9. When you close the application the Citrix session disconnects and you can save your recorded script.

BDL scripts of recorded Citrix Web Interface sessions are multi-protocol scripts that include a small number of Silk Performer Web functions.

Citrix Script Functions

- `CitrixWaitForWindowCreation`, used for screen synchronization, is the most important Citrix function.

  The first parameter that synchronizations consider is window caption. If during replay a window caption is returned that matches the caption of the window that was recorded during replay, then the verification succeeds.

  Note: Wildcards (*) can be specified at the end or beginning of window captions.

  The second parameter considered is the match parameter. Normally this is an exact, case-sensitive comparison of window caption strings.

The third parameter is the style of the window. If a window caption name is not available, then a portion of the style name is used to identify the window. Styles reflect whether or not windows were maximized during recording.

The fourth parameter considered is the position of the window. This may be a negative number, as maximized windows have a position of `-4, -4`. This parameter can be controlled via the **Force window position** profile setting. When this profile setting is enabled, windows are automatically moved to the exact positions they held during recording.

The fifth parameter is window size. During replay windows must be resized to the same size they had during recording.

- `CitrixWaitForLogon` waits until a Citrix client logs on successfully or a specified timeout period expires.
- `CitrixWaitForScreen` captures screen regions and checks them against specified conditions (normally a hash value captured at recording). If a condition does not match and the timeout period expires, the function call fails. `CitrixWaitForScreen` functions also use screen appearance to synchronize subsequent user actions. Based on provided parameters, this function waits until the image in a specified screen region changes, matches the hash value captured at recording, or does not match the hash value captured at recording.
- `CitrixGetScreen` takes a screenshot of a selected region and writes the screenshot to a file in the result directory. If the file name is omitted, it will be automatically generated by the user ID and hash value of the image.
- `CitrixGetScreenHash` retrieves the hash value of a selected screen.
- `CitrixSetOption` sets particular options, such as network protocol specification, speed screen latency reduction, data compression, image caching, mouse/keyboard timings and event queueing, client disconnect, synchronization time-outs, think times, TrueLog capture, and window position forcing.
- `CitrixWaitForWindow` waits until a specified window event (specified with the `nEvent` parameter) occurs. Such events may be an activation, destruction or caption change for a specified window. If the selected event is a caption change, a matching caption change for the specified window will satisfy the function. Captions can be specified explicitly using the `sCaption` and `nMatch` parameters.
- `CitrixKeyString` is the standard function for entering printable characters.
- `CitrixMouseClick` moves the mouse to a specified position and presses a specified button. Optionally, the mouse can be specified to move while the button is pressed. Key modifiers (such as `Ctrl` and `Alt`) can be used.

Screen Synchronization and Verification

Silk Performer supports bitmap and window verification for applications that are hosted by Citrix XenApp servers. Screen synchronization offers a means of verifying replayed Citrix content. Screen synchronizations differ from standard script verifications in that they allow for verification of window and screen region appearances, not input values. Also, they are inserted during script recording, not during subsequent script customization. Screen synchronizations are particularly useful for synchronizing subsequent user input that is displayed in browsers, or similar interfaces (for example, Citrix application windows), because such applications do not support automatic synchronization through window events such as user input and text display.

Note: Response data verification is not supported for Citrix testing.

Silk Performer relies on hash values to verify replayed bitmaps against recorded bitmaps. Hash values are computer-readable values that reflect bitmap specifications such as size, position, resolution, and color depth.

Note: To verify replay screen regions against hash values that are captured at recording it is necessary that the same color depth that is used during recording be used during replay. Scripts fail

when these specifications are not maintained because changes as small as a single pixel can change hash values and result in replay content appearing to be different from recorded content.

Note: Windows maximized during recording must also be maximized during replay. This is because replay cannot change the state of windows (it can only resize and move windows). So if a window state changes (for example, from `Maximized` to `Restored`), then it is likely that some user input in the script caused the change (for example the **Restore** button may have been clicked). On replay the user will click at the same screen position (now the **Maximize** button) and consequently a different operation will be executed and the subsequent `CitrixWaitForWindowRestore` function will fail.

Text and Screen Synchronizations

Window synchronizations such as `CitrixWaitForWindow()` and `CitrixWaitForWindowCreation()` are well suited to synchronizing with an application. It is important to synchronize with the application so that the script waits until the application is ready for additional user input. If you do not use synchronizations, the script will most likely overrun the application. Also, synchronization gives you point-in-time reference as to when tasks are completed, effectively measuring response times.

Text Synchronization

Many tasks that can be performed on an application do not show, hide, create, or destroy windows, they simply change a section of the screen. In such instances you should use the Silk Performer text synchronization functionality.

After recording your script, you can visually add text synchronization points via the TrueLog Explorer `Synchronize Text` option. Text synchronization works via built-in OCR (Optical Character Recognition).

Screen Synchronization

Silk Performer offers two types of screen synchronizations: `wait for content change` and `wait for content match`. This form of synchronization requires a window reference and two sets of x/y coordinates. However since unplanned events can lead to displaced window coordinates, you should use text synchronization whenever possible, and only use screen synchronization when there is no text to compare.

Wait for content change waits until the selected portion of the screen changes, while wait for content match waits until the selected portion of the screen matches what was defined during recording.

Note: Screen synchronization points must be inserted while you record your test case, whereas text synchronization points can be inserted afterward, via TrueLog Explorer.

Generating a Screen Region Synchronization During Recording

Screen synchronization is achieved via `CitrixWaitForScreen` functions, which are not scripted automatically by the recorder. These functions are inserted via the **Screen Region** dialog box during recording. `CitrixWaitForScreen` functions compare replay and record bitmaps to determine whether or not they are identical. Hash values, as opposed to actual bitmaps, are used to compare the images. This limits resource consumption during replay.

Note: Screen region synchronization is only available via the Silk Performer Citrix Recorder.

1. Record a Citrix session and create a test script.
2. During recording, click **Select Region** on the Silk Performer Citrix Recorder.
3. Click and drag your cursor to select the screen region for which you want to generate a bitmap synchronization.

Note: Because differences as small as a single pixel can cause synchronization processes to fail, it is recommended that for text verifications you select the minimum screen area required. Otherwise unanticipated screen differences (for example, disabled toolbars) may affect verification results.

4. The **Selection** dialog box opens. Specify how you want to have screen region coordinates scripted (`Script absolute coordinates`, `Script coordinates relative to window`, or `No coordinates, use full window`). When windows are maximized there is effectively no difference between absolute and relative coordinates. When windows are not maximized, relative coordinates are measured from the top-left corner of the Citrix Recorder window, while absolute coordinates use fixed x/y coordinates.
5. Specify the **Content matching type** that the Citrix XenApp player should wait for during replay (`content match`, `content mismatch`, or `content change`).
6. Click **OK** to add the synchronization to your Citrix XenApp test script.

Verification and Parsing via OCR

Silk Performer support for optical character recognition (OCR) simplifies session-dependent verifications and parsing by recognizing text values in the screengrabs of captured application states.

Verification and parsing functions are added via TrueLog Explorer after script recording.

Window Position and State

Window position and state (maximized/minimized) is important for insuring accurate replay as TrueLog Explorer scripts screen coordinates where selected text is to be read from relative to the desktop, not individual windows. So if a window appears in a different position during replay than it did during recording, OCR operations will not be able to locate the specified text. If it is not possible to specify an absolute position, the script must be manually updated using coordinates relative to windows.

Adding OCR Verification Functions

With OCR support Silk Performer enables the storing of recognizable text values in variables, thereby simplifying session-dependent verifications in Citrix tests.

Overview

String verification via optical character recognition (OCR) is achieved using `CitrixVerifyText` API calls. These functions are inserted via TrueLog Explorer during script customization. `CitrixVerifyText` functions compare text strings in replay bitmaps to determine if they are identical.

`CitrixParseText` functions are available for parsing text. These API calls work in the same way as other Silk Performer parsing functions.

Optical character recognition relies on pattern databases to recognize varying fonts and text styles. Font databases must be generated before OCR can be run.

Citrix TrueLogs show verification and parsing API calls in the tree view.

Note: OCR operations must be performed on stable content because when working with frequently changing screen images replay timing is critical. When synchronizing on window events it is possible that screen refresh may be slightly delayed, which will result in timing dependent outcome. Therefore it is good practice to either script a wait or a `CitrixWaitForScreen` function call before each OCR verification/ parsing function.

The following two screen examples show the output of verification and parsing functions after TryScript runs.

Generating an OCR Verification Function

1. From Silk Performer record a Citrix session
2. Run a TryScript run with the **Animation** checkbox selected on the **TryScript** dialog box. This opens TrueLog Explorer.
3. When the TryScript run is complete, select a `CitrixSynchronization` API node (or child node) that includes a bitmap screengrab of a page on which you want to verify text.
4. Click and drag your cursor onscreen to select the page region that includes the text you want to use for verification.
5. Right-click in the selected area and select **Verify Selected Text** from the context menu.
6. The **Insert Text Verification Function** dialog box opens. The selected text is pre-loaded into the constant value edit box and the constant value radio button is selected by default.

   In addition to being able to verify against a constant value, you can also verify against an existing parameter or a new parameter. To verify against a parameter, select the parameter radio button. If a parameter already exists, clicking "**...**" enables you to browse to and select the parameter. If no parameters exist, clicking "**...**" launches the **Parameter Wizard**, which you can use to create a new parameter.
7. From the **Verify that the text in the selected rectangle is** drop list, select `equal` or `not equal`.
8. Specify whether or not the verification is to be **Case sensitive** and should **Ignore whitespaces**.
9. In the **Severity** portion of the dialog box, specify the severity that is to be raised if the verification returns a negative result (`Error`, `Warning`, `Informational`, or `Custom`).
10. Click **OK**.
11. A confirmation dialog box appears. Click **OK** to add the OCR verification function to your Citrix test script.

Generating an OCR Parsing Function

1. From Silk Performer record a Citrix session
2. Run a TryScript run with the **Animation** checkbox selected on the **TryScript** dialog box. This opens TrueLog Explorer.
3. When the TryScript run is complete, select a `CitrixSynchronization` API node (or child node) that includes a bitmap screengrab of a page from which you want to parse text.
4. Click and drag your cursor to select the page region that includes the text you want to parse.
5. Right-click in the selected region and select **Parse Selected Text into a Variable**.
6. The **Insert Parsing Function** dialog box offers parameters by which the parsing function can be configured. Though the default settings will likely be correct, you can adjust:

   Parameter name - Enter the name of the parameter that is to receive the result of the parsing function.

   Informational statement insertion - Select **Print statement** to insert an informational Print statement into the script after the Web page call. This writes the result of the parsing function to the Silk Performer **Virtual User Output** window.

   Select **Writeln statement** ("write line" statement) to write the parsed value to an output file to facilitate debugging (in addition to writing the value to the **Virtual User Output** window as a Print statement does). Because generating output files alters test measurements, these files should only be used for debugging purposes and should not be generated for full tests.
7. Click **OK**.
8. A confirmation dialog box appears. Click **OK** to add the OCR parsing function to your Citrix test script.

Trying Out a Generated Script

With TryScript runs only a single virtual user is run and the stress test option is enabled so that there is no think time or delay between transactions.

Note: Although Citrix TrueLogs do not include live display of data downloaded during testing (via TrueLog Explorer) Citrix Web Interface TrueLogs do include live display of downloaded data. Both TrueLog types include the writing of log files, report files, and replay within the Silk Performer Citrix Player.

If you have configured parsing or verification functions based on Citrix OCR support, you must generate an OCR font database before attempting a TryScript run, otherwise these functions may not operate correctly.

1. Click **Try Script** on the Silk Performer Workflow bar. The **Try Script** dialog box opens.
2. To view live display of page content within TrueLog Explorer during replay (Citrix Web Interface TrueLogs only), select the **Animated Run with TrueLog Explorer** check box.

   The Visible Citrix Client option (Citrix TrueLogs only, not available for Citrix Web Interface TrueLogs) enables visual replay in the Silk Performer Citrix Player during TryScript runs.
3. Click **Run**.

   Note: You are not running an actual load test here, only a test run to see if your script requires debugging.
4. The TryScript run begins. The Silk Performer Monitor window opens, giving you detailed information about the run's progress.

Citrix TrueLogs

The Silk Performer Citrix Player open for Citrix TryScript runs. TrueLog Explorer opens for Citrix Web Interface TryScript runs (when the **Animation** checkbox on the TryScript dialog box is checked). TrueLog Explorer displays the data that is actually downloaded during TryScript runs.

By selecting a high-level synchronization node you see a bitmap of the window captured during replay just as it appeared after the last synchronization function.

Window synchronization functions are visualized with colored borders. Window creations are indicated with green borders. Window activations are indicated with blue borders. Window destructions are indicated with yellow borders.

TrueLogs work in complement with the Silk Performer Citrix Player by visualizing screen states. For example, if you are not sure which window is indicated by a certain window ID that is listed in the Silk Performer Citrix Player Log window, you can find the corresponding synchronization function in the corresponding TrueLog and thereby access a bitmap that shows the window.

User input nodes (`CitrixUserInput` and related functions) reflect keyboard and mouse input. `CitrixMouseClick` functions offer two track vector parameters (X and Y coordinates). Red diamonds indicate mouse-click start points. Red cross-marks indicate mouse release points. A red line between a start and end point indicates the path of the mouse. If there is no move while the button is pressed, then only a red cross is displayed. Onscreen tool tips offer additional information (for example, `right-click`, `left-click`, `double-click`).

Value strings (keyboard input) are visualized onscreen as floating red text until target window captions are identified (in subsequent nodes) to indicate where strings are to be input.

Silk Performer Citrix Player

The Silk Performer Citrix Player opens when TryScript runs begin, replaying all recorded actions in full animation. Mouse movements and operations are simulated with an animated mouse icon.

Note: Silk Performer Citrix Player only opens for the Citrix application type, not the CitrixWeb Interface application type.

Click **Toggle Log Window** in the upper right corner of the player to open the **Log** window. The **Log** window includes three panes that detail different aspects of TryScript runs:

- **Script** - This pane lists all of the executed BDL script functions and the currently executing BDL function.
- **Windows** - This pane includes a stack of all the client windows of the current session, including window captions, styles, sizes, and positions. Top-level windows carry a window icon and are listed above sub-windows.
- **Log** - This pane lists all informational messages and events, including executed BDL functions, and window creation, activation, and destruction.

In all panes, active functions and windows are indicated with a blue arrow icon.

Step by Step Replay

1. During a Citrix TryScript run, click the **Toggle Log Window** button in the upper right corner of the Silk Performer Citrix Player to open the **Log** window.
2. Click the **Step** button. Replay stops at the active function. A blue arrow icon indicates the next function in the script.
3. Click **Step** to execute the next function.
4. Continue clicking **Step** to advance through the script.

   Note: You can also enable step-by-step execution by selecting the **Step by step execution** checkbox on the **TryScript** dialog box.
5. Click **Run** to resume continuous script processing.

Skipping Time-Outs

The Silk Performer Citrix Player waits for all time-outs that are encountered during replay. To avoid waiting for time-outs (default time-out is 60 seconds), click **Skip** to advance past them.

Note: Clicking **Skip** generates user-break errors in scripts.

1. During a Citrix TryScript run, click the **Toggle Log Window** button in the upper right corner of the Silk Performer Citrix Player to open the **Log** window.
2. When replay encounters a time-out, click **Skip** to advance to the next function.

Customizing User Data

With user data customization you can make your test scripts more realistic by replacing static recorded user input data with dynamic, parameterized user data that changes with each transaction. Manual scripting is not required to create such data-driven tests.

During testing you can customize the user input that is entered into applications that are hosted by Citrix terminal services in two ways:

- The **Parameter Wizard** allows you to specify values to be entered with keyboard events, enabling your test scripts to be more realistic by replacing recorded user input data with randomized, parameterized user data.
- Visual customization allows you to customize mouse events such as clicks, drags, and releases.

Using the Parameter Wizard

1. Select an API node that reflects user data input (for example, select a `CitrixKeyString` node that specifies a keyboard datastring).
2. Right-click the input datastring (shown as floating red text) and select **Customize User Input** from the context menu.
3. The **Parameter Wizard** opens. Select **Create new parameter** and click **Next**.
4. With the **Parameter Wizard** you can modify script values in one of two ways. You can either use an existing parameter that's defined in the `dclparam` or `dclrand` section of your script, or you can create a new parameter (based on either a new constant value, a random variable, or values in a multi-column data file). Once you create a new parameter, that parameter is added to the existing parameters and becomes available for further customizations.

   

   Note: This task explains only the process of creating a parameter based on a new random variable.
5. The **Create New Parameter** dialog appears. Select the **Parameter from Random Variable** radio button and click **Next**.
6. The **Random Variable Wizard** appears. From the drop list, select the type of random variable (for example, `Strings from file`) you want to insert into your test script. A brief description of the highlighted variable type appears in the lower window.
7. Click **Next**.
8. The **Name the variable and specify its attributes** page appears. The `Strings from file` random variable type generates data strings that can either be selected randomly or sequentially from a specified file. Enter a name for the variable in the **Name** field. Specify whether the values should be called in `Random` or `Sequential` order. Then select a preconfigured datasource (for example, `Elname` which defines last names) from the **File/Name** drop list.
9. Click **Next**.
10. The **Choose the kind of usage** page appears. Specify whether the new random value should be used `Per usage`, `Per transaction`, or `Per test`.
11. Click **Finish** to modify the BDL form declaration of your test script so that it uses the random variable for the given form field in place of the recorded value. The new random variable function appears below in BDL view.
12. Initiate a TryScript run with the random variable function in your test script to confirm that the script runs without error.

Customizing Mouse Events

1. Select a `CitrixMouseClick` node that includes mouse activity. Red diamonds indicate mouse-click start points. Red cross-marks indicate mouse-release points. A red line between a start and end point indicates the path of the mouse. Onscreen tooltips offer additional information (for example, `right-click`, `left-click`, and `double-click`).
2. Click anywhere on the screen and select **Customize User Input** from the context menu. The **Customize Mouse Event** dialog box appears.
3. Click at the screen position where you want the customized mouse move to begin.
4. Click at the screen position where you want the customized mouse move to end.
5. Click the **Customize** button to accept the customization and modify the BDL script accordingly.

   Your mouse event customization now appears in the recorded TrueLog bitmaps in green. The mouse customization also appears in the BDL script in green text. `CitrixMouseClick` functions offer two track vector parameters (X and Y coordinates). The next time this script executes, it will use the new screen coordinates you have specified.

Synchronizing Text

TrueLog Explorer offers a synchronization function that pauses the execution of Citrix functions until specified text or a text pattern appears in a specified location. Such synchronization is vital for the accurate execution of verification functions.

Project and System Settings

Citrix profile settings are project-specific settings related to Citrix synchronization, logging, virtual user simulation, and client options. Citrix settings are specified on a per-project basis.

This section focuses on Citrix replay settings. Citrix record options are limited to network protocol, encryption level, the `Log screen before each user action` setting, and the `Use RAM disk` setting.

Configuring Citrix XenApp Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, scroll down to and click the **Citrix** icon. The **General** tab opens.
4. In the **Synchronization timeout** text box, type a default synchronization timeout in milliseconds that is to be used by all `CitrixWaitForXXX` functions that do not specify a timeout value.

   The default value is `60000 ms`.
5. Check **Force window position** to make the calls to `CitrixWaitForWindowCreation` and `CitrixWaitForWindowRestore` move windows to the coordinates captured during recording (enabled by default).

   When this check box is not checked, both the `CitrixWaitForWindowCreation` and `CitrixWaitForWindowRestore` functions provide the parameter `bForcePos` to enable this option for each individual call.
6. Check **Disconnect on transaction end** to disconnect the Citrix client following the end of each transaction, including the `TInit` transaction (disabled by default).
7. Check **Gracefully disconnect session** to perform a log-off when disconnecting from a Citrix XenApp session (enabled by default).
8. Check **Log screen before each user action** to capture and write a screenshot at the beginning of each user action (enabled by default).

   When TrueLog generation is enabled, a screenshot is taken at the end of each synchronization function and written to the TrueLog.

   The `CitrixWaitForScreen` function captures a screen region and checks it against a specified condition instead of against a hash value that is captured at recording. This file can later be examined or compared to what was captured during recording for error analysis.

   If the function call `CitrixWaitForScreen` fails and **Dump window region on unsuccessful screen synchronization** is checked , the captured screen region is written to a file in the result directory. The function call `CitrixWaitForScreen` may fail, for example, if the condition does not match when the timeout period expires.

9. Check **Use RAM disk** to use a different drive for the intermediate storage of images.

   Note: If you check **Use RAM disk**, you must select the appropriate drive letter of the RAM disk from the list box. TrueLog generation performance improves if the specified drive is a RAM disk.

10. Select the **Simulation** tab.
11. In the **Length of time mouse button remains pressed** field, type the length of time that the virtual user is to hold the mouse button in the pressed state.

    The default value is `200 ms`. The functions `CitrixMouseClick` and `CitrixMouseDblClick` use this value.
12. In the **Length of time between the clicks of a double-click** field, type the maximum length of time that can pass between the two clicks of a double-click.

    The default value is `100 ms`. The function `CitrixMouseDblClick` uses this value.
13. In the **Mouse speed** field, type the speed (in pixels per second) at which the mouse is to move across the screen.

    The default value is `1000 pixels`.
14. In the **Length of time each key remains pressed** field, type the length of time that the virtual user must hold a keyboard key in the down state.

    The default value is `50 ms`. The functions `CitrixKey` and `CitrixKeyString` use this value.
15. In the **Length of time between keystrokes when entering strings** field, type the length of time that must pass between the individual keystrokes.

    The default value is `100 ms`. The function `CitrixKeyString` uses this value.
16. In the **Key repeat time** field, type the time required for a complete key stroke when simulating repeat functionality.

    The default value is `50 ms`. A value of `50 ms` represents 20 keys per second.
17. In the **Delay after successful synchronization** field in the **Think times** area of the tab, type the length of time that virtual users are to remain inactive after passing a successful synchronization point.

    The default value is `1000 ms`. The function `CitrixWaitForXXX` uses this value.
18. In the **Delay after each user action** field, type the length of time that virtual users remain inactive between actions.

    The default value is `100 ms`.
19. Click the **Citrix client** tab to specify Citrix client options.
20. From the **Network protocol** drop box, select the low-level network protocol to use for locating and connecting to the Citrix server.

    For more information, refer to Citrix client documentation.
21. Check the **Use data compression** check box to compress all transferred data (enabled by default).

    This feature reduces file size but requires additional processor resources.
22. Check the **Use disk cache for bitmaps** check box to store commonly used graphical objects, such as bitmaps, in a local disk cache (disabled by default).
23. Check the **Queue mouse movements and keystrokes** check box to queue mouse and keyboard updates (disabled by default).

    This feature reduces the number of network packets sent from the Citrix client to the Citrix XenApp server.
24. From the **SpeedScreen latency reduction** drop box, select one of the following to enhance user experience on slower network connections:

    - For WANs and other slower connections, select `On`.
    - For LANs and other faster connections, select `Off`.
    - To turn latency reduction on and off based on the latency of the connections, select `Auto`.
25. From the **Encryption level** drop box, select a level of encryption for the ICA connection.

    The Citrix XenApp server must be configured to allow the selected encryption level or higher.

Note: Using an encryption level other than the server default or `Basic` disables automatic logon to the Citrix XenApp server.

26. Click **OK** to save your settings.

Configuring General Settings

1. In the Project tab of the tree-view area of the main SilkPerformer window, right-click the Profiles node and select Edit Active Profile.
2. The Profile - [Profile1] - Simulation dialog opens at the Simulation tab (Replay category).
3. Scroll down to and select the Citrix icon. The Citrix General settings tab opens.
4. In the **Options** section of the **General** tab, specify a timeout value in the **Synchronization timeout** field. You may have to increase this value if your Citrix server is slow.
5. The `Force window position` option (enabled by default) automatically moves replayed windows to the coordinates specified in `CitrixWaitForWindowCreation` functions.
6. The `Disconnect on transaction end` option (disabled by default) disconnects the client after each transaction, even init transactions.
7. In the **Logging** section of the **General** tab, the `Log screen before each user action` option (enabled by default) enables onscreen display of user input (datastring values) for the moment before values are actually input into screens (for example, a user might enter a string value into a spreadsheet cell. The value is not actually input until the user pushes **Enter**. With this option enabled, in the node just preceding the click of the **Enter** key, the string value appears onscreen as floating red text). This option requires significant processing and disk storage as it dictates that each user action generate a screengrab. With this option disabled you do not see all user input updates.
8. The `Dump window region on unsuccessful screen synchronization` option specifies that screengrabs be generated for all unsuccessful screen synchronizations. These bitmaps, when captured and saved to the current result directory (for example, `RecentTryScript`) of your Silk Performer installation, can be compared to corresponding recorded synchronizations to assist in debugging efforts. For example, a difference of a single pixel is enough to cause a screen synchronization error. Such a difference might best be detectable visually, by comparing recorded screengrabs with screengrabs captured when synchronization errors occur.
9. The grabbing, reading, compressing, and writing of screengrabs for TrueLogs involves significant processing resources and can lead to slow replay. The `Use RAM disk` option (disabled by default) enables faster TrueLog replay by making use of a RAM disk or solid state drive (SSD), rather than writing files to a conventional hard disk. Use the drop list to select the letter of a drive with high read/write rates. Note that this option does not install a RAM disk, but you may use the RAM disk of your choice.
10. Click **OK** to save your changes, or click **Default** to restore the default settings.

Configuring Simulation Settings

1. In the **Project** menu tree, right-click the **Profiles** node and select **Edit Active Profile**.
2. The **Profile - [Profile1] - Simulation** dialog opens at the **Simulation** tab (Replay category).
3. Scroll down to and select the Citrix icon.
4. Select the **Simulation** tab.
5. In the **Mouse** section of the **Simulation** tab, specify virtual user mouse behavior (in milliseconds) such as the length of time that mouse clicks remain pressed, the length of time between the clicks of a double click, and mouse speed. Note that simulated mouse events move at constant speeds. They do not simply jump across the screen.
6. In the **Keyboard** section of the **Simulation** tab, specify virtual user keyboard behavior (in milliseconds) such as the length of time that keys remain pressed, the length of time between keystrokes when

entering strings (for example, `CitrixKeyString` functions), and key repeat time (irepeat parameters of `CitrixKeyString` functions).

7. In the **Think times** section of the **Simulation** tab, specify virtual user think time behavior (in milliseconds) for delay after successful synchronizations, and delay after each user action. This is a virtual simulation of user reaction time that helps to stabilize replay.
8. Click **OK** to save your changes, or click **Default** to restore the default settings.

Configuring Client Settings

1. In the **Project** menu tree, right-click the **Profiles** node and select **Edit Active Profile**.
2. The **Profile - [Profile1] - Simulation** dialog box opens at the **Simulation** tab (Replay category).
3. Scroll down to and select the Citrix icon.
4. Select the Citrix client tab.
5. Select the network protocol upon which your client will run (`TCP/IP` or `TCP/IP + HTTP`). When you specify `TCP/IP + HTTP`, load balancing is done with the HTTP protocol, using post commands. When you specify `TCP/IP`, UDP is used. No other network protocols are supported.
6. Check the **Use data compression** checkbox to enable data compression (enabled by default).
7. Check the **Use disk cache for bitmaps** checkbox to enable the caching of bitmaps on your hard disk (disabled by default).
8. Check the **Queue mouse movements and keystrokes** checkbox to queue mouse movements and keystrokes for a specified period of time before they are sent to the server (disabled by default).
9. **SpeedScreen latency reduction** enables local echo of mouse and keyboard actions (disabled by default). **Local echo** means that you do not have to wait for round trips to the server to see the results of your input. Specify `Off`, `On`, or `Auto`.
10. Specify an **encryption level** for the client. Options include, `Use server default`, `Basic`, `128 Bit for Login Only`, `40 Bit`, `56 Bit`, and `128 Bit`.
11. Click **OK** to save your changes, or click **Default** to restore default settings.

Enabling Citrix Replay on Agents

1. From the Silk Performer menu bar, select **Tools** > **System Configuration Manager** . The **System Configuration Manager** dialog box appears.
2. *(applicable to all Citrix XenApp client versions)* Click the **Applications** tab. Click the Silk Performer icon. Enter **Account** and **Password** credentials for a user account on the agent workstation that has permission to execute Citrix sessions.
3. To record a Citrix published application that is accessed using the settings defined on the Citrix Recorder **Connect** dialog box, perform the following:
   a) Expand the **Profiles** node on the Project menu tree and right-click the active profile. Select **Edit Profile**.
   b) Scroll down to and click **Citrix**.
   c) Click the **Citrix client** tab.
   d) From the **Network protocol** list box , select `TCP/IP + HTTP`.

   To record a Citrix published application utilizing an ICA file: In the `[WFClient]` section of the ICA file, change `TcpBrowserAddress=` to `HttpBrowserAddress=`.

System Settings for OCR

To enable optical character recognition (OCR) for parsing and verification functions in TrueLog Explorer, you must generate a font database using Silk Performer system settings.

Optical character recognition relies on font (or pattern) databases to recognize fonts and text styles in bitmaps. The default set of fonts covers most scenarios, however in some situations you may want to add

additional fonts or font styles. A new database should be generated whenever new fonts are added or removed from the system.

Including too many fonts in the database can slow down processing and lead to contradictory reading, so it is recommended that you only include those fonts that are used in the bitmaps from which you will be capturing text strings.

Configuring System Settings for OCR

1. Within Silk Performer, go to `Settings/System.../` and select the Citrix icon.
2. On the **OCR** tab, use the **Add >>** and **Add All** buttons to move those fonts that you wish to have used for OCR from the **System Fonts** list box to the **Chosen Fonts** list box.
3. Use the **Remove All** and **<< Remove** buttons to delete unnecessary fonts from the **Chosen Fonts** list box.
4. In the **Sizes** field, specify the font size range that should be used (for example, `8-20`).
5. Define which font styles should be included by selecting the `Italic`, `Bold`, and `Underlined` check boxes. Make sure you select the `Underlined` check box if you want to verify menu entries.
6. Click **Generate Font Database**.
7. The **Build Font Base** dialog box opens. Click **OK** to confirm that you want to replace the existing font database with a new database.

   

   Note: If you encounter problems while generating the font database, it is likely a data execution prevention issue. To resolve the issue, please see the section below.
8. Click **OK** on the Silk Performer System Settings dialog box to accept the changes.

Testing Best Practices

While GUI-based testing presents a number of challenges, it can be rewarding as it offers the opportunity to closely simulate a real user experience. GUI-based testing through Silk Performer involves applying load to applications by simulating the terminal services protocol.

At least two physical computers should be used to perform Citrix tests: one computer runs the terminal services environment and the application under test (AUT); the other computer runs Silk Performer and the Citrix client software.

The load generator computer simulates a large number of users by mass producing the terminal services network protocol. The terminal services server (system under test) simulates multiple MS Windows-based desktop sessions simultaneously. This server also hosts the GUI application that is placed under load.

We recommend that you place an isolated network (LAN) between the load generator and the system under test. This allows for network throughput analysis. An isolated network is also less vulnerable to external influence, thereby reducing errors and misleading network throughput results.

Test Preparation

- User Interface (UI) design - Take advantage of any opportunity to influence the UI design of the application. A solid, consistent UI allows you to create reusable sub-routines. In turn, you can write less test code, and consequently have less code to maintain.
- Test plan - A thoughtful test plan is a prerequisite of successful GUI-based load testing. In your test plan you should define your goals, objectives, test runs, critical metrics, and others.
- Use cases - Use cases are step-by-step scenarios that you plan to test. Use cases represent typical user-to-application interaction, for example starting an application, opening a project, editing settings, and others. Your use cases guide you through the recording process.

- Application expert - Even when equipped with well documented use cases you may not be able to make sense of all of an application's workflow. It is helpful to have access to someone who has expert knowledge of the application under test, for example a business process owner.
- Screen resolution - The higher the screen resolution, the higher the system requirements are per terminal session. We recommend you use a screen resolution of no more than 800x600 for record and replay. This approach allows for compatibility with older computers, which you then have the opportunity of including in your load tests.
- Remote users - If your application is to be available to remote users, consider simulating a slow network during some of your load tests. Slow network connections can have a major impact on the performance of applications, therefore additional tests with simulated network congestion may help improve your test results. You can use the network emulation functionality of Silk Performer for this purpose.

Defining Your Goals

The first thing to do when planning your load test is to define your goals. While testers often assume that their goals are clear to all involved parties, they often are not. It is important that the quality assurance team and management work toward the same goals.

When writing up goals, first consider your highest-level goals:

1. Discover the baseline performance of your application
2. Optimize/tune your application
3. Determine if the application under test is ready for production

Once you have established your high-level goals, you need to break them down into clear, measurable objectives, covering issues such as how you plan to accomplish your goals, how you plan to measure your goals, and what results will be considered acceptable.

Here is an example of how you might break down your testing goals:

Goal #1: Discover and document the baseline performance of your application

- Use Silk Performer to measure the response times of critical application functions.
- Place timer functions around window/screen synchronizations (`WaitFor` events). These measurements will show up in the final report.

Critical timers:

- Measure how much time it takes from when the **OK** button is clicked on the **Login** window until the **Welcome** window appears.
- Measure how much time it takes from when the **Query results** button is clicked until the populated list is displayed.

Goal #2: Optimize/tune the application

- Optimize/tune the application for 5 days.
- Document all changes and their impact on the performance of the application under test.

Goal #3: Determine if the application under test is ready for production

- The application is ready for production if the following condition is met: All response times are under 5 seconds (Silk Performer provides metrics for each window and screen sync).

With a list of measurable objectives, your test results define a definite point at which the application will be ready for production. This also eliminates the risk of endlessly optimizing the application as tuning it to the defined goal is adequate.

Creating Use Cases

A use case is a typical task that a user undertakes when working with the application under test. A use case must use the features of the application that require testing. It is essential that you only test features that are important and working properly. This is not the time to perform functional testing, which should

already have been completed. Testing is a long process: the longer the use cases, the more time that will be required for testing.

When stepping through a use case, write down all significant screen events. For example, when entering a formula in Microsoft Excel you should document the changing of cells due to formula processing. Such events will translate into screen synchronizations that will be important during script development. Text synchronizations can be used for text-based screen synchronizations.

Document use cases to a detailed level. You need to document every mouse click, key stroke, and expected result. While this may be tedious initially, it makes things easier when you begin recording your test cases.

The following example, a test that simply locates an existing instance of Microsoft Word and opens a document, displays the level of detail that a use case should have. The square brackets (“[ ]”) indicate an event.

```
In the “Microsoft Word” window, navigate to the File menu and select Open….
[The “Open” dialog box opens]
Select ‘Test.doc.’
Click Open.
[The “Open” dialog box closes]
[The “Microsoft Word” window has focus once again]
```

Note that the exact titles of the windows are documented in the above example. This becomes important later during script development as such information is needed to keep the script in sync with the application. Once you have a well documented use case, it is easy to script the application

There are several ways to write use cases. You can use XML notation, numbered notation, or another format that you are familiar with.

XML Use Case Example

```
<Task Name=”Open a document”>
  In the “Microsoft Word” window, navigate to the File menu        and select
Open….
  [The “Open” dialog box displays]
  Select ‘Test.doc.’
  Click Open.
  [The “Open” dialog box closes]
  [The “Microsoft Word” window has focus once again]
</Task>
```

Numbered Use Case Example

```
100.01 – Open a document
  In the “Microsoft Word” window, navigate to the File menu        and select
Open….
  [The “Open” dialog box displays]
  Select ‘Test.doc.’
  Click Open.
  [The “Open” dialog box closes]
  [The “Microsoft Word” window has focus once again]
100.02
    …
```

Ensuring a Stable Environment

Before you start recording, make sure that there will be no additional changes to the UI of the application under test, at least until your tests are complete. This requires good communication with the development team. If the UI changes after you record your test scripts, your scripts will likely be unusable.

Also, make sure that you have the ability to back up and restore any databases that are used in your testing. Because changes to database content often lead to differences in UI content, for example more or

different items in lists, different query results, and other, databases need to be returned to a baseline state before the start of each test cycle.

Troubleshooting

Once you have documented your use cases, recording or coding your test scripts should be a relatively simple task. With Silk Performer you have both the record/playback option and the option of coding your test scripts manually. Each approach has its advantages and disadvantages. The most efficient way to create test scripts with Silk Performer is to combine the two approaches, as follows:

1. With Silk Performer, record the scenario you have outlined in your use case description.
2. Use TrueLog Explorer to visually customize your script (this is where the “Text synchronization” feature comes into play).
3. For more complex scripting, edit the generated BDL script manually.

   Manually editing a script is also the best option for inserting the content of your use case into your test script (in the form of comments). See the example below, which uses numbered notation format:

```
100.001 In the “Microsoft Word” window, navigate the menu to File, Open….
100.002 [The “Open” dialog window shows]
100.003 Select Test.doc.
100.004 Click Open.
100.005 [The “Open” dialog window goes away]
100.006[The “Microsoft Word” window has focus again]
```

4. You can now copy/paste these contents into your BDL script as follows:.

```
// 100.001 In “Microsoft Word" window,
    hwndWordMainWindow := CitrixSearchWindow(“*Microsoft
    Word”, MATCH_Wildcard);
    CitrixWindowBringToTop(hwndWordMainWindow);

// navigate the menu to File, Open….
    CitrixKey(KEY_Alt);
    CitrixKeyString("f");
    CitrixKeyString("o");

// 100.002 [The “Open” dialog window shows]
    hwndOpenDialog := CitrixWaitForWindowCreation(“Open”,
    Match_Exact);

// 100.003 Select Test.doc.
    CitrixMouseClick(150, 100, hwndOpenDialog, MOUSE_
    ButtonLeft);

// 100.004 Click Open
    CitrixMouseClick(300, 200, hwndOpenDialog, MOUSE_
    ButtonLeft);

// 100.005 [The “Open” dialog window goes away]
    CitrixWaitForWindow(hwndOpenDialog, EVENT_Destroy);

// 100.006 [The “Microsoft Word” window has focus again]
    CitrixWaitForWindow(hwndWordMainWindow, EVENT_Activate);
```

   If the script fails, these comments will help you determine where the script failed in reference to the use case.

Recording Tips

When recording a terminal services script, you must be aware that unplanned events can change the position and size of target windows during replay, causing test runs to fail. Here are some general tips that will help you avoid the effort of finding and fixing such replay errors.

Try launching the application via the Windows **Run** command. Launching the application via a desktop icon or mouse click may also work, but consider if you will be moving the application to a different server, or if the GUI will vary between users, in which case a desktop icon might appear in a different location. It is therefore recommended that you type the path to the application's EXE file in the Windows **Run** command window.

Keyboard Shortcuts

Mouse clicks require x/y coordinates, a window reference, and the target window must have focus. While GUI testing technologies for tracking mouse clicks have become increasingly reliable, Windows remains a dynamically changing environment, so you must build scripts that can tolerate minor inconsistencies. One way to do this is to use keyboard shortcuts. For shortcuts, the only requirement is window focus.

Note: The **Windows Logo** key is not supported, as it moves the focus away from the Citrix Recorder.

Almost all menu systems in Windows applications have **Alt** key combinations that can be used to initiate specific menu commands. To record use of an **Alt** key shortcut, first give your application focus, then press the **Alt** key, this gives the menu system focus. Notice that many of the menu items have one of their letters underlined. This indicates which **Alt** key commands are available for specific menu items. Press appropriate letter keys to gain access to sub-menus until you reach the menu item that you want to initiate. In some cases, a menu item itself will have a key combination. For example, to open a document in Microsoft Word use the `Ctrl+O` combination. This shortcut does not require that the menu have focus.

Maximizing Windows

If keyboard shortcuts are not available and you must use the mouse, you can reduce the chance of triggering inaccurate mouse clicks by maximizing the window that you are working with. This places the window in a fixed position each time it is used and, in turn, makes mouse clicks more accurate. As a general rule, if a window allows for maximization, it is recommended that you take advantage of it.

Note: `Alt-Space, X` is the Windows keyboard combination for maximizing an active window

When a window does not allow for maximization, during the recording session, move the window to the upper left-hand corner of the desktop before clicking. When recording, the recorder may not be able to pick up on the window handle that you have in the foreground, so having the coordinates relative to the desktop is a must.

Keeping the System Clean

To eliminate unexpected events during terminal services sessions, ensure that you keep your Windows desktop as uncluttered as possible. Imagine, for example, that while your script is replaying, a Windows network message appears. Your script will break because the subsequent mouse click will go to the Windows network message window instead of your intended application window.

Ensure Correct Focus

Before clicking a window, ensure that the window you intend to click exists and has focus. Here is a sample BDL function that helps automate this process:

```
function MyCitrixWaitForWindowCreationAndActivation(sCaption:string;
                nMatch    : number optional;
                nStyle    : number optional;
                nX        : number optional;
```

```
                nY        : number optional;
                nWidth    : number optional;
                nHeight   : number optional
                ) : number
var
    hwndNewWindow  : number;  // hwnd of the new window created
    nRTT:number;
begin
    //
    // Check if window exists
    //
        hwndNewWindow := CitrixSearchWindow(sCaption, nMatch);
          hwndNewWindow := 0;
    //
    // If window doesn't exist, wait for creation
    //
        if (hwndNewWindow < 1) then
            hwndNewWindow := CitrixWaitForWindowCreation(sCaption, nMatch,
            nStyle, nX, nY, nWidth, nHeight);
        end;
    //
    // Check if window is already active, wait if it's not active
    //
        MyCitrixWaitForWindowActivate(hwndNewWindow);
        MyCitrixWaitForWindowCreationAndActivation:=hwndNewWindow;

end MyCitrixWaitForWindowCreationAndActivation;
```

Using Parameters

If you input the same information more than once, the system under test caches the data in memory rather than repeating the same work. For this reason it is important that you vary input/request data so that the work is as realistic as possible. For example, use different user accounts that have different data ranges. This can be accomplished using a CSV (comma separated values) file as input for a test.

Adding Verifications

Verifications are checks that are added to code to check if the responses that are received from the server are correct. When performing GUI based testing, verifications are automatically added to your script via window synchronizations, however it is recommended that you add additional verifications to your code.

Adding Timers

Custom timers are critically important for tracking an application's response times. Without custom timers, you cannot determine your end user's overall experience with an application. Metrics can help determine which aspects of your application are performing well and which are not.

Before you begin adding custom timers to your test script, identify the critical areas of your application and determine where your `MeasureStart` and `MeasureStop` calls should be placed. For this effort it is good practice to review the log in TrueLog Explorer.

Here is an example of using timers in a Silk Performer script:

```
//
// Submit a work order.
//
CitrixMouseClick(27, 31, hwndWorkOrder, MOUSE_ButtonLeft);

//
// Start the response time clock.
//
MeasureStart("202.01: Work Order Submission.");

//
```

```
// Wait for the "Order Submission Complete" dialog box.
//
MyCitrixWaitForWindowCreationAndActivation(
    "Order Submission Complete",
    MATCH_Exact
    );

//
// Stop the response time clock.
//
MeasureStop("202.01: Work Order Submission ");
```

Debugging Scripts

Windows may fail to be activated and screen synchronizations may fail when Silk Performer Citrix replay encounters different values during replay than were captured during recording. Sometimes the causes of synchronization problems are not apparent. They may be due to a change in screen position of only a single pixel.

More common than screen synchronization failures are windows not being activated during replay. In such cases the screenshots associated with the corresponding user actions may explain the fault.

Note: Sometimes there is no user fault and a window is activated only sporadically. In such cases you must remove the associated `CitrixWaitForWindow` function.

Silk Performer Citrix replay captures screengrabs when errors occur (the default setting) and writes the bitmaps to disk. By default, the recorder writes screenshots to the screenshots directory in the project directory. Replay stores screenshots in the current used result directory. Visual comparison of record and replay screens is best achieved using bitmap viewing programs.

The Silk Performer **Dump window region of unsuccessful screen synchronizations** Citrix option must be activated (the default) to have bitmaps captured and saved.

Troubleshooting Scripts

You may encounter timeout errors or other execution failures during load test execution. Here are some tips for avoiding, finding, and fixing such errors.

Using TrueLog Explorer

Test runs often fail due to the appearance of unexpected dialogs, causing scripts to lose focus on the windows they are working on. It is therefore recommended that you enable the TrueLog On Error feature in Silk Performer before you execute tests. Then, if an error occurs, you will be able to visually track it down in TrueLog Explorer.

Clearing Terminal Server Sessions

After a failed test execution, ensure that you reset all terminal server sessions before you execute the test again, otherwise your script will likely fail.

Handling Application Errors

During tests, your application is likely to throw errors due to generated load. Adding event handlers to your script that can handle and report such errors is very helpful.

Here is an example of an event handler that continuously watches for a window with the name **Program Error Intercepted** and executes an `ALT-C` to close any such window that appears. The event handler also generates an error message whenever such an error occurs.

```
dclevent
  handler Handler1 <EVENT_CITRIXINTERRUPT>
  var
```

```
    nInterrupt, nWindow : number;
    nStyle              : number;
    sWindowCaption      : string;
    begin
      CitrixGetActInterrupt(nInterrupt, nWindow);
      ErrorAdd(FACILITY_CITRIXENGINE, 47, SEVERITY_INFORMATIONAL);
      print(string(nWindow));
      CitrixGetWindowCaption(nWindow, sWindowCaption);
      if sWindowCaption = "Program Error Intercepted" then
        CitrixKey(67, MOD_Alt); // 'c'
      end;
      ErrorRemove(FACILITY_CITRIXENGINE, 47);
    end Handler1;
```

Avoiding Think Times

While it may be tempting to use `Wait()` or `ThinkTime` statements to avoid having scripts overrun applications under test, this practice is not recommended for two reasons:

- When load generation increases, application processing speed may slow considerably. The wait statement may eventually be too short and the problem of overrunning the application will again present itself.
- If you are measuring the response times of window, text, or screen synchronizations, the time in the `Wait()` statement will artificially bloat response times.

The solution is to use synchronizations.

Re-recording Portions of a Script

Sometimes changes in application behavior result in portions of a recorded script becoming obsolete. Re-recording a portion of a use case is an option, but beware that the recorder may not be able to track the handles of the existing windows, resulting in incorrect window handle numbers. These issues can make the process of integrating newly recorded script with an original script quite tedious. When a use case is small, it is recommended that you re-record the entire use case. Otherwise, follow the process outlined below to integrate a new script with an outdated script:

1. Comment out the section of code that is to be re-recorded.
2. Match the location of the code section requiring replacement with the corresponding section in the use case.
3. Bring a terminal services session up to the corresponding point in the use case.
4. Begin recording the open terminal services session.
5. Perform the actions of the use case that require replacement.
6. Stop the recorder (a script from the recording is then produced).
7. Replace the window handle variables in the newly recorded script with the respective handles of the original script. You may use functions such as `CitrixSearchWindow()` to locate correct window handles.
8. Copy the edited code into the original script.

Handling Citrix Dialog Boxes

Depending on how you connect to Citrix terminal services sessions and your licensing setup, you may see one or both of the following dialog boxes:

- ICA Seamless Host Agent
- Citrix License Warning Notice

These dialog boxes are informational and may or may not appear when you initially log into terminal services sessions. Following are two ways of handling these dialog boxes.

Handling Citrix dialog boxes (Solution #1)

This solution creates an interrupt that handles the dialog boxes if they appear:

```
transaction TMain
  var
  begin
    CitrixInit(800, 600);
    CitrixAddInterrupt(INTERRUPT_WindowCreate, "ICA Seamless Host Agent",
MATCH_Exact);
    CitrixConnect("lab74", "labadmin", "labpass", "testlab1", COLOR_16bit);
    CitrixWaitForLogon();
    hWnd4 := CitrixWaitForWindowCreation("", MATCH_Exact, 0x96840000, -2,
572, 804, 30);
    CitrixWaitForWindowCreation("Program Manager");
    CitrixMouseClick(36, 17, hWnd4, MOUSE_ButtonLeft, MOD_None, -1, 0);
    hWnd11 := CitrixWaitForWindowCreation("", MATCH_Exact, 0x96400000, 2,
313, 163, 263);
    CitrixMouseClick(62, 247, hWnd11, MOUSE_ButtonLeft);
    CitrixWaitForWindow(hWnd11, EVENT_Destroy);
    hWnd12 := CitrixWaitForWindowCreation("Shut Down Windows", MATCH_Exact,
0x94C808CC, 191, 136, 417, 192);
    CitrixWaitForWindow(hWnd12, EVENT_Activate);
    CitrixMouseClick(203, 170, hWnd12, MOUSE_ButtonLeft);
    CitrixWaitForDisconnect();
  end TMain;

dclevent
  handler Handler1 <EVENT_CITRIXINTERRUPT>
  var
    nInterrupt, nWindow : number;
    nStyle              : number;
  begin
    CitrixGetActInterrupt(nInterrupt, nWindow);

    ErrorAdd(FACILITY_CITRIXENGINE, 47, SEVERITY_INFORMATIONAL);
    print(string(nWindow));
    if CitrixGetWindowStyle(nWindow, nStyle) and (nStyle <> 0xB4000000) then
      CitrixWaitForWindow(nWindow, EVENT_Activate);
      CitrixMouseClick(201, 202, nWindow, MOUSE_ButtonLeft);
      CitrixWaitForWindow(nWindow, EVENT_Destroy);
    end;
    ErrorRemove(FACILITY_CITRIXENGINE, 47);

  end Handler1;
```

Handling Citrix dialog boxes (S #2)

This sample code loops for 30 seconds, waiting for the Citrix dialog boxes to appear. If the dialog boxes appear, this code closes them.

```
function MyCitrixStartup(nMaxWait: number optional): boolean
    var
        hwndICAHandle              : number;
        hwndFoundLicenseWarning    : number;
        nCount                     : number;
    begin
        hwndICAHandle:=-1;
        hwndFoundLicenseWarning:=-1;
        nCount:=0;

        if (nMaxWait = 0) then
            nMaxWait:=10;
        // if no wait time was passed,
        // use 10 tries (seconds) as a default
        end;
```

```
        MeasureStart("MyCitrixStartup");

    //
    // loop until we've handled the conditions or we've tried
    //

        while ((nCount < nMaxWait) and ((hwndICAHandle <=0) or
            (hwndFoundLicenseWarning <=0))) do

    //
    // Just a little feedback, every 10 tries
    //
            if ((nCount MOD 10) =0) then
                print(string(nCount) + ")MyCitrixStartup "
                + " vUser:" + string(GetUserId())
                + " hwndICAHandle=" + string(hwndICAHandle)
                + " hwndFoundLicenseWarning="
                + string(hwndFoundLicenseWarning),
                OPT_DISPLAY_ERRORS , TEXT_GREEN );
            end;
    //
    // if we haven't handled this window yet
    //
            if (hwndICAHandle  <=0)
then
                hwndICAHandle := CitrixWaitForWindowCreation
                ("ICA Seamless Host Agent", MATCH_Exact, 0x94C800C4,
                0, 0, 0, 0, false, 1, true);

                if (hwndICAHandle > 0) then
                    if (CitrixWindowBringToTop(hwndICAHandle)) then
                        CitrixKey(KEY_ENTER); // press ok to close the dialog
                        CitrixWaitForWindow(hwndICAHandle, EVENT_Destroy);
                        // wait for the close
                    end; // end waiting for window to top
                end; // end if we have a valid handle
            end; // if window has not been found yet

            if (hwndFoundLicenseWarning  <=0)
then
                hwndFoundLicenseWarning := CitrixWaitForWindowCreation
                ("Citrix License Warning Notice", MATCH_Exact, 0x94C800C4,
                0, 0, 0, 0, false, 1, true);

                if (hwndFoundLicenseWarning > 0) then
                    if (CitrixWindowBringToTop(hwndFoundLicenseWarning)) then
                        CitrixKey(KEY_ENTER);  // Press ok
                        CitrixWaitForWindow(hwndFoundLicenseWarning,
                            EVENT_Destroy);
                        // wait for it to go away
                    end; // end waiting for window to top
                end; // end if we have a valid handle
            end; // if window has not been found yet

            nCount :=nCount+1;
            Wait 1.0;
        end; // while nCount

        MeasureStop("MyCitrixStartup");
        //
        // return true if we handled any one of these conditions
        //
```

```
        MyCitrixStartup2 := (hwndFoundLicenseWarning > 0)
            or (hwndICAHandle > 0) ;

          print("MyCitrixStartup "
            + " vUser:" + string(GetUserId())
            + " Waited " + string(nCount) + " of " + string(nMaxWait)
            + " hwndICAHandle=" + string(hwndICAHandle)
            + " hwndFoundLicenseWarning=" + string(hwndFoundLicenseWarning),
            OPT_DISPLAY_ERRORS , TEXT_GREEN );

end MyCitrixStartup;
```

Index

T

U

W