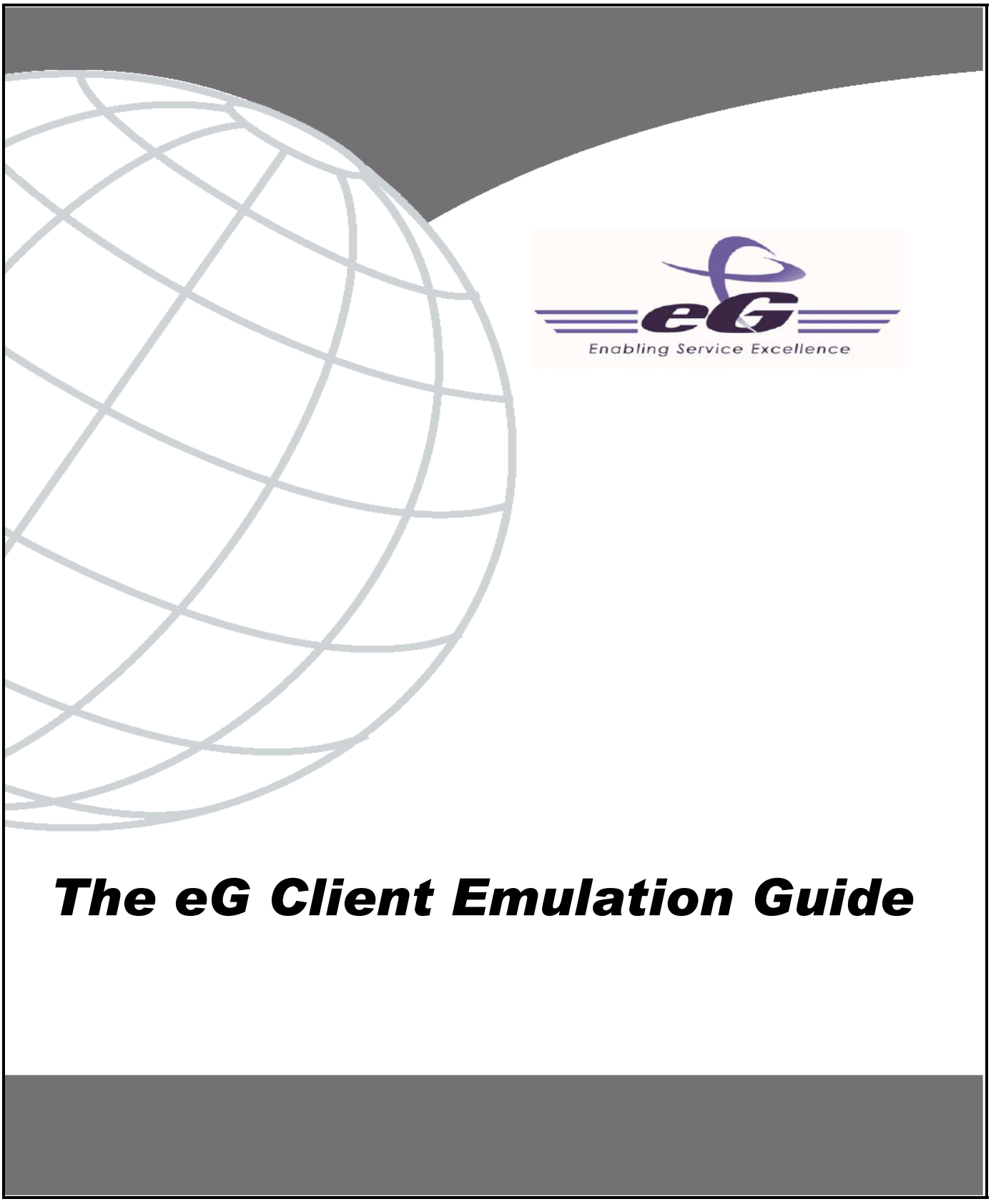

The eG Client Emulation Guide

Restricted Rights Legend

Trademarks

Copyright

Table of Contents

Table of Figures

Chapter 1

Introduction

As IT infrastructures evolve into being business-critical, high availability and peak performance of the IT infrastructure is becoming as critical as reliability is to a telephone network. In such infrastructures, it is imperative to determine in real-time when service failures or slowdowns problems occur, and to accurately pin-point which step(s) in the service access are impacting the end-user experience. Speedy problem detection and accurate diagnosis can reduce service downtime and the consequent business impact of critical IT services.

eG's client emulation and monitoring capability goes well beyond basic protocol-level up/down testing. IT administrators can record typical user accesses to mission-critical infrastructure services, and later have eG agents periodically playback the recorded accesses using the exact same client applications that users employ to access a service. The eG client emulator effectively simulates multi-step user interactions with a service (e.g.,login, browse, submit data, fill-in forms, etc.).

By collecting availability and response time statistics for the complete service interaction and comparing the metrics with time-of-day, auto-generated or administrator-defined fixed thresholds, the eG client emulator immediately alerts administrators during potential service outages or slowdowns.

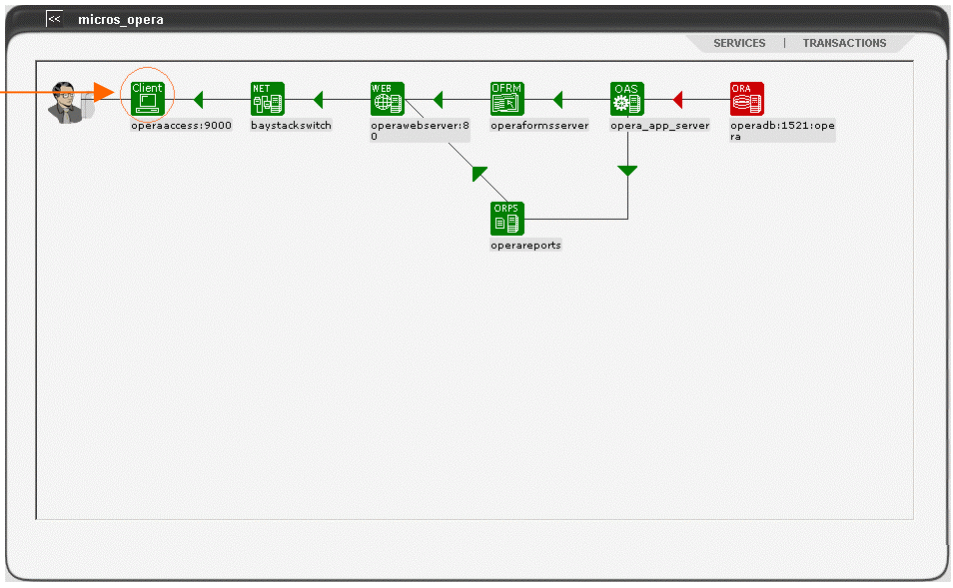

Figure 1.1: A service topology representation comprising of an emulated client component

By comparing the response times across each of the steps of a multi-step service access, administrators can detect the exact step(s) that could be causing user-visible slowdowns. Furthermore, using its patented auto-triage capability, the eG suite is able to correlate the end-to-end service performance indicators provided by the eG client emulator with critical indicators of network, system, and application availability, performance, and usage to pin-point where the root-cause of a slow-down lies. Customers can use the integrated solution to improve the quality of their service offerings, thereby enhancing their competitive positioning, lowering their operations costs, and optimizing the usage of their infrastructure.

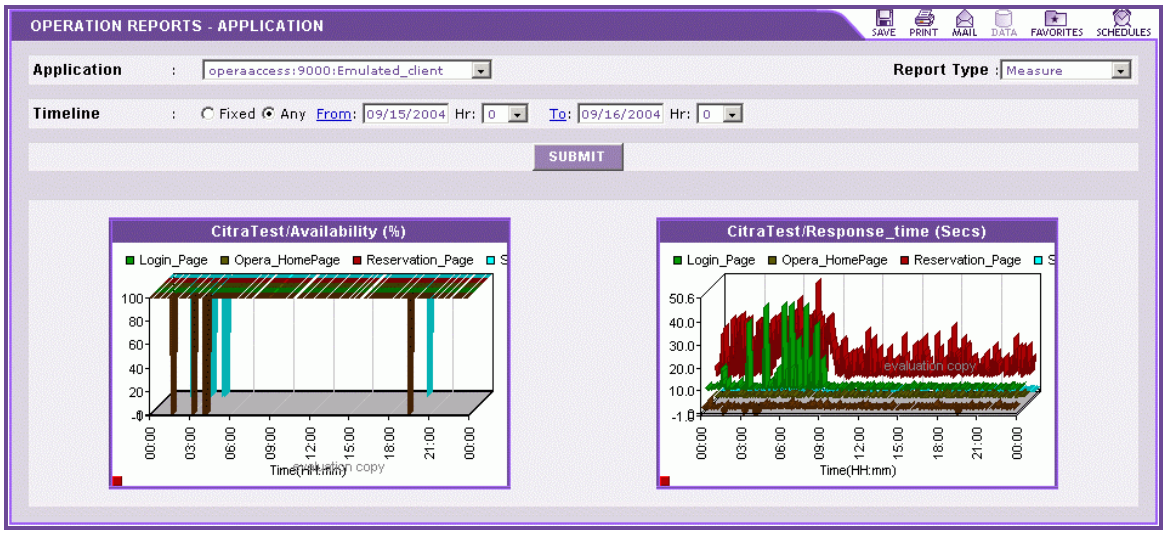

Figure 1.2: Comparing the availability and response times for each step of a service interaction

Since it is capable of supporting web-based or native client applications, the eG suite can handle web-based as well as proprietary/native client applications (and even thin-client/Citrix applications) equally well.

1.1 Benefits of the eG Client Emulator

- Emulate real user interactions to Web/non-Web services and record service availability and end-to-end response times;
- Identify which step(s) of a service interaction are causing slowdowns;
- Provide instantaneous alerts on service outages and slowdowns;
- Faciliate root-cause diagnosis and infrastructure optimization through correlation with other network, server, application performance indicators;
- Offers a reliable, cost-effective way of automating routine service health checks;

1.2 Architecture

Typically, the first step in using the eG client emulator is to record a sequence of user activities when accessing the service(s) to be monitored. The end result of the recording process is a script that can be later played back to emulate user activity. The recorded script not only includes the sequence of clicks necessary to emulate the user activity, but also has information on checkpoints - each checkpoint corresponds to a step in the multi-step user interaction (e.g., logging in, adding to shopping cart, etc.).

To record the script, eG integrates with one of the two client emulation tools: Tevron's CitraTest, and Seapine Software's QA Wizard. A development environment for the client emulation tool is necessary to perform the recording.

The recorded script can be played back on a dedicated system, and the results are reported as availability and response time measures for the overall playback activity. In addition, response times of each of the steps involved can be obtained so as to assist the IT service manager in pin-pointing which of the steps of the multi-step user interaction with the service could be causing a slow-down of the overall service.

The integration of the eG suite with the client emulation tools is performed at the eG agent side. An eG agent is provided with the location of the recorded script that it has to execute to emulate a user activity. Based on the pre-specified frequency of the test, the agent executes the script, analyzes the script results, and reports availability and response time information to the eG manager. The eG manager receives these client emulation reports and correlates them in real-time with critical in-depth resource usage and server-side processing metrics that it receives from the other agents, to report on potential bottlenecks in the target IT infrastructure.

The eG external agent implements the integration with the client emulation tools. Since the client emulation tools require a dedicated system to operate on, an external agent can either perform client emulation tests or the other eG protocol emulation tests, but not both. When adding a new external agent, depending on the eG license available, the administrator can specify whether the agent must be allowed to perform client emulation activity or not.

Figure 1.3 depicts how the integration of the eG external agent with the client emulation tools works. The recorded script file must be made available (this is a manual process) on the system that the eG external agent is running on. Also, for the agent to execute the recorded script, the runtime environment of the client emulation tool must be installed.

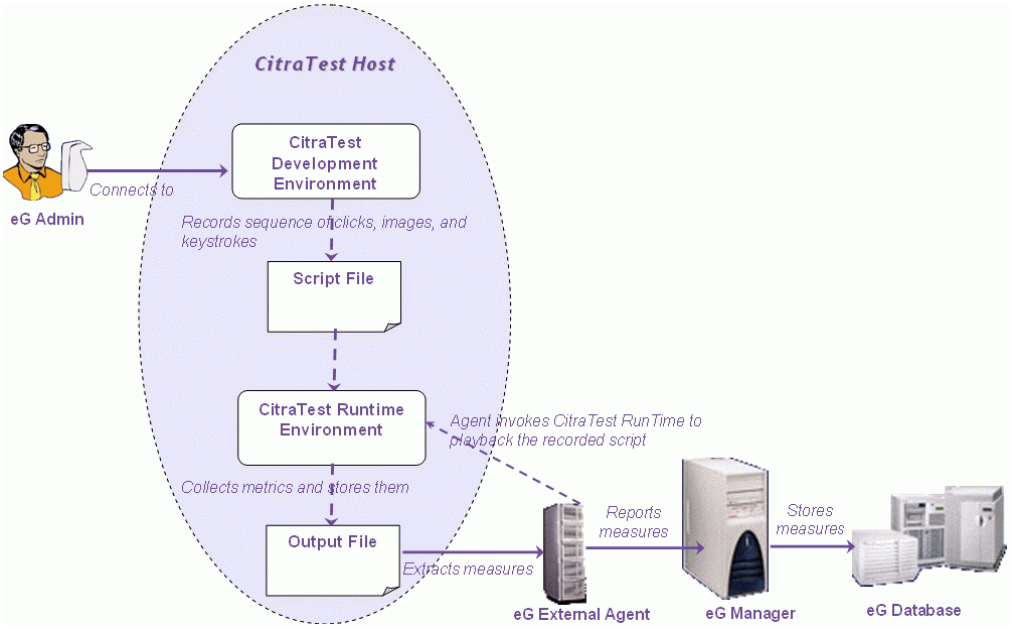

Figure 1.3: A typical eG-Client emulation tool Integration

1.3 Licensing

- An eG external agent is required for monitoring emulated clients
- eG external agents assigned to handle emulation tests cannot execute other eG tests
- Client emulation is supported on Windows platforms only
- For the integration to work, the eG license should have "Client Emulation" capability enabled
- The Client emulation tool must be installed when an eG external agent with client emulation is started
- The Client emulation tool's development environment is mandatory for building a script

The chapters that follow will discuss how eG is integrated with each of the Client emulation tools i.e., CitraTest and QA Wizard.

Chapter 2

Integrating eG Enterprise with CitraTest

CitraTest is an automated performance testing tool for Web and non-web-based Windows applications. This tool emulates a user request to an application by recording the sequence of visuals, mouse clicks, and keystrokes that constitute a typical "user access", in a VB script file. When this script is executed, the recorded sequence is replayed, and the availability and response time of the application are measured.

2.1 Steps for Integrating eG with CitraTest

The primary steps involved in the eG-CitraTest integration are installing the CitraTest Development environment and building a script file using it. To ensure a smooth and effective integration, **CitraTest v5.0 or v6.06** needs to be installed. For downloading the CitraTest installable, refer to the URL: http://www.tevron.com/download.asp, For an elaborate installation procedure for CitraTest, refer to the *CitraTest manuals*.

Note:

Since CitraTest emulates real user transactions by opening client applications and emulating user clicks on the client application, CitraTest requires complete control of the desktop of the system on which it executes. Therefore, CitraTest should be installed on a **dedicated system**.

The key prerequisites for installation include:

Software Prerequisites:

- Windows NT or Windows 2000 or Windows XP server/workstation
- Microsoft Internet Explorer (IE) version 5.0 or higher
- Microsoft Visual Basic 6.0 (This is required only for installing and using the CitraTest development environment. For the CitraTest Runtime environment though, the VB runtime files that are bundled with the Windows operating system will suffice; in such a case, Microsoft VB is not required).

Others:

- Screensavers should be disabled on the system executing CitraTest
- Use the **LockMachine.exe** available in the **<CITRATEST_INSTALL_DIR>** to disable the keyboard and mouse of the workstation on which CitraTest is executing. For more details pertaining to the **LockMachine** utility, refer to the *CitraTest Manuals*.

Once the development environment is installed, proceed to record a test script. A test script is a sequence of actions that are recorded as a user accesses one of the services in the target infrastructure. The recorded script can then be played back to emulate user accesses to the service. Image and text recognition techniques are used during playback to determine whether playback of a script succeeded or not.

After building the script file, use the eG administrative interface to configure the script playback. Next, start the external agent which will playback the script, and finally, view the measures returned by CitraTest in the eG monitor interface.

Note:

If CitraTest is installed and the script created on a remote host using a normal Remote Desktop session, then the eGurkhaAgent service will not be able to execute the CitraTest script due to lack of the requisite permissions. On the contrary, if all the above-mentioned processes had been implemented using the '/console session' of the Remote Desktop, then the eGurkhaAgent service will execute the script and report measures to the eG manager.

The sections to come will take the help of two illustrated examples to explain how eG client emulation works. While the first example deals with a web application, the second example targets a Citrix application.

2.2 Client Emulation for a Web Application

The script file that will be built in the first example will emulate a user request to the **Flight Status** page of the **Singapore Airlines** web site hosted by the web server 192.168.10.32 on port 80. Using the **Flight Status** page the availability information of a specified flight is retrieved. The example will be used to monitor critical transactions along the way.

2.2.1 Building a Script File

To build the script file, do the following:

1. Open the CitraTest Development environment using the menu sequence depicted by Figure 2.1.

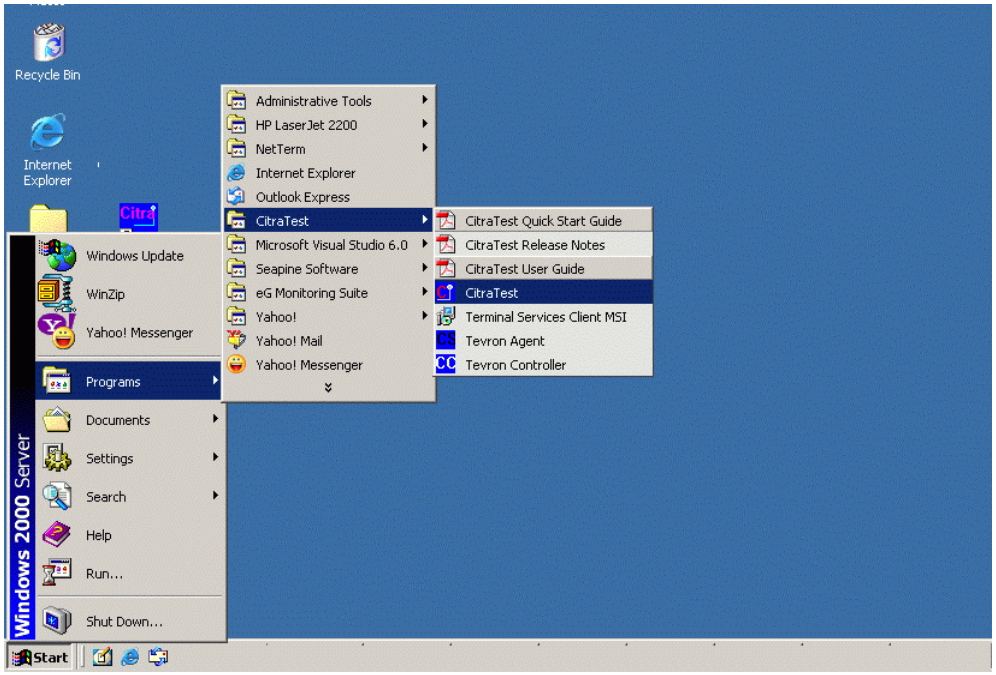

Figure 2.1: Opening the CitraTest console

2. To create a new script, next, select the **New** option from the **File** menu of Figure 2.2.

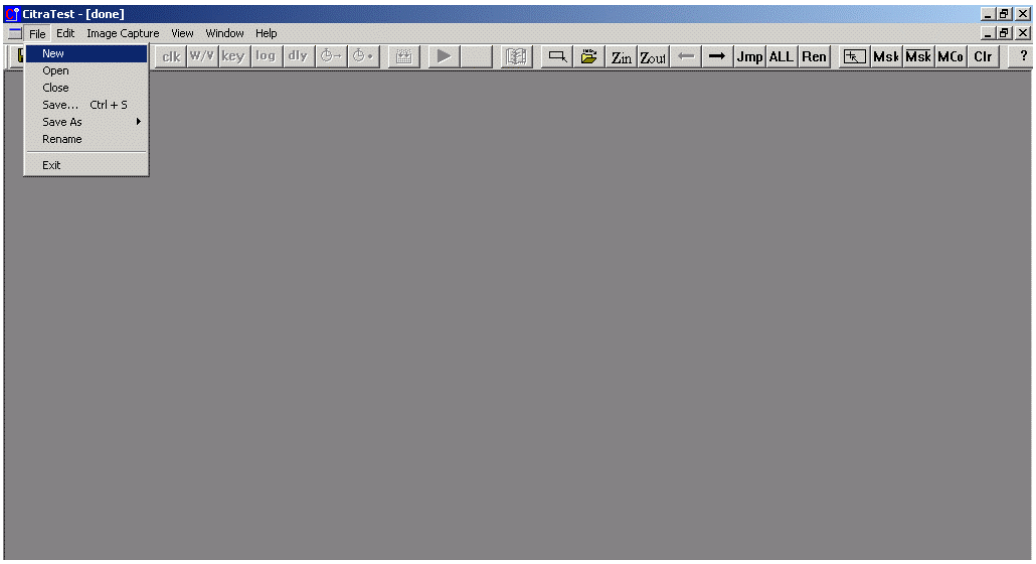

Figure 2.2: Creating a new script

3. In the next screen (see Figure 2.3), a **Script Type** and **Development Type** need to be selected. The **VU Load Test script** is used for stress testing applications. For measuring the availability and responsiveness of an application, the default **Standard Script** has to be used. Therefore, select the **Standard Script** option from Figure 2.3. Then, select **VisualBasic 6.0 Project** as the **Development Type** and finally, click the **OK** button in Figure 2.3.

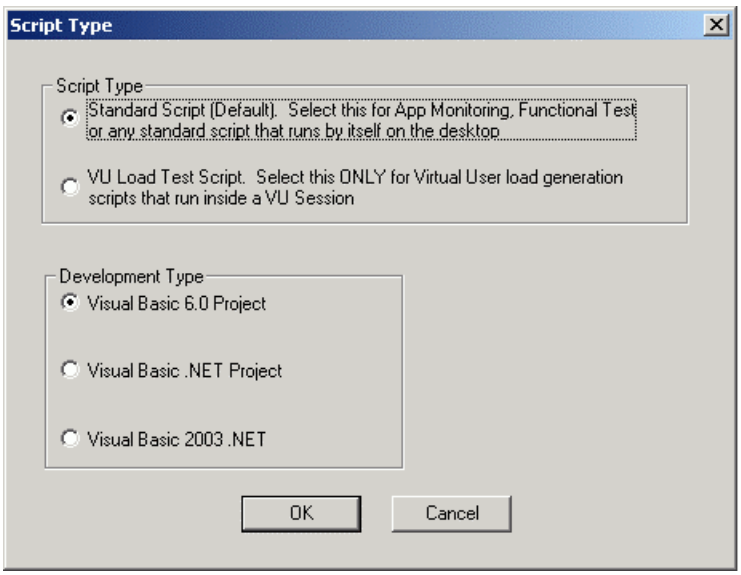

Figure 2.3: Selecting a Script type and a Development type

4. Next, Figure 2.4 will appear prompting you to specify the name of the new VB script that is being created, and the directory to which it should be saved. Let us name the VB script in our example as **SAWebSite**. Specify this name against the **File name** text box of Figure 4, and click the **Open** button therein.

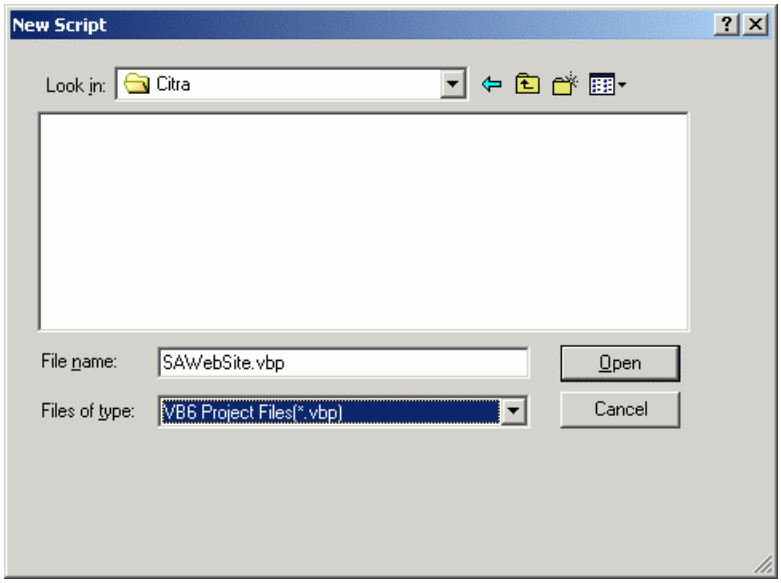

Figure 2.4: Providing a name and location for the new VB project

5. Next, specify the directories to which the images to be captured and the search areas to be defined need to saved. A **Search Area** is defined when a specific area in a page needs to be searched for the existence of an image. By default, the images and search areas will be stored in the same directory as the VB script file. You can change the location for the images and search areas, if need be, using Figure 2.5. A default directory for fonts will also be displayed in Figure 2.5. All the fonts that are registered with the displayed directory will alone be recognized by CitraTest. You can set a different font directory by

specifying a different path in the **Set Font Directory** text box. The buttons adjacent to every text box can be utilized to browse for a particular location.

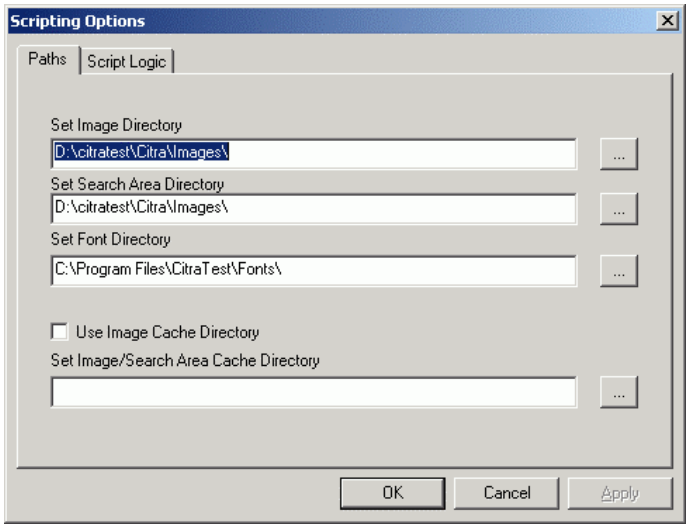

Figure 2.5: Specifying the path to the images and search areas

6. Clicking on the **Script Logic** tab of Figure 2.5 will display the default termination logic (see Figure 2.6) for the VB script. During playback, if any of the conditions specified in the VB script fails, then the displayed script will automatically run and abort the script execution. If desired, you can modify this script.

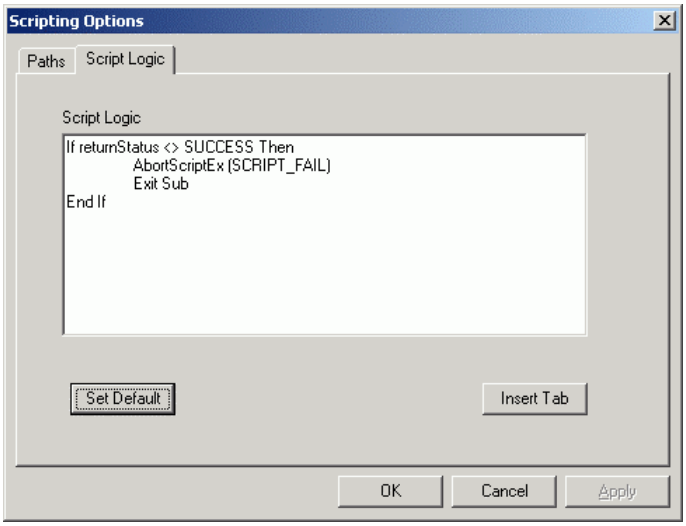

Figure 2.6: The default termination logic

7. Once the path and script changes are made, click on the **OK** button in Figure 6 to proceed with the script creation.
8. The script window will then open (see Figure 2.7).

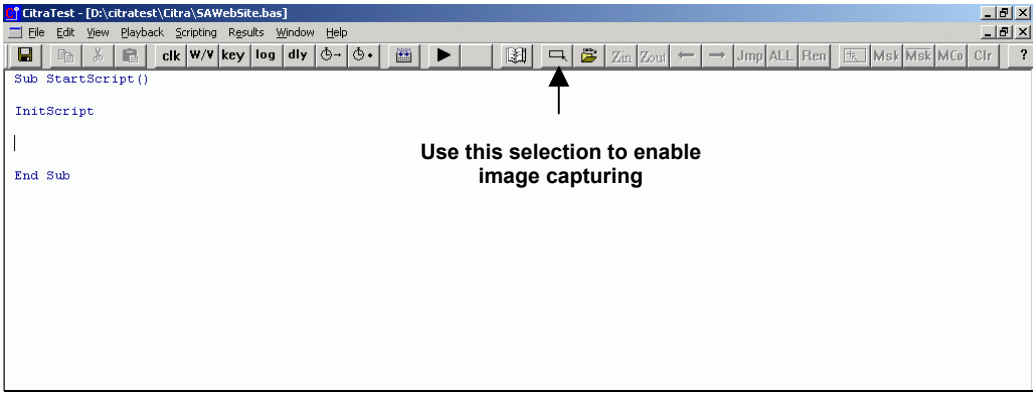

Figure 2.7: The script window

9. When a VB script file is created, an output file with the same name as the script file will also be created by default, in the script file directory. This output file has the extension **`.timer.csv`**. Therefore, the default output file that corresponds to the script file in our example will be **`SAWebSite.timer.csv`**. This output file will store the availability and response time values, which will later be extracted by the eG agent and reported to the eG manager.

10. Next, begin capturing the images, clicks, and key strokes that form part of the user transactions that are to be emulated. Before commencing image capturing, it would help if you prepare a list of broad steps that need to be followed to access the application to be monitored. These broad steps can then be broken down into individual clicks and key strokes. This exercise will help you identify the shortest and the most efficient sequence of steps that will result in a valid user request.

    Since our example involves accessing the **Singapore Airlines** web site, the first step would be to open the Internet Explorer. This, in turn, involves restoring the Windows desktop and then clicking on the **Internet Explorer** icon on it.

    To restore the Windows desktop, you need to click on the **Restore Desktop** button on the Windows taskbar. To record this click, first, capture the **Restore Desktop** button image. To achieve this, first, enable image capturing by clicking on the  button on the tool bar of Figure 2.7.

11. Once image capturing is enabled, the mouse cursor will change to resemble a 'plus' (+) sign. Now, switch to the Windows desktop, place the '+' cursor on the left top corner of the **Restore Desktop** button, click there using your left mouse button, and then, with the left mouse button pressed drag the cursor until it covers the entire **Restore Desktop** button (see Figure 2.8). As you drag, a dashed line will appear indicating the area that has been covered.

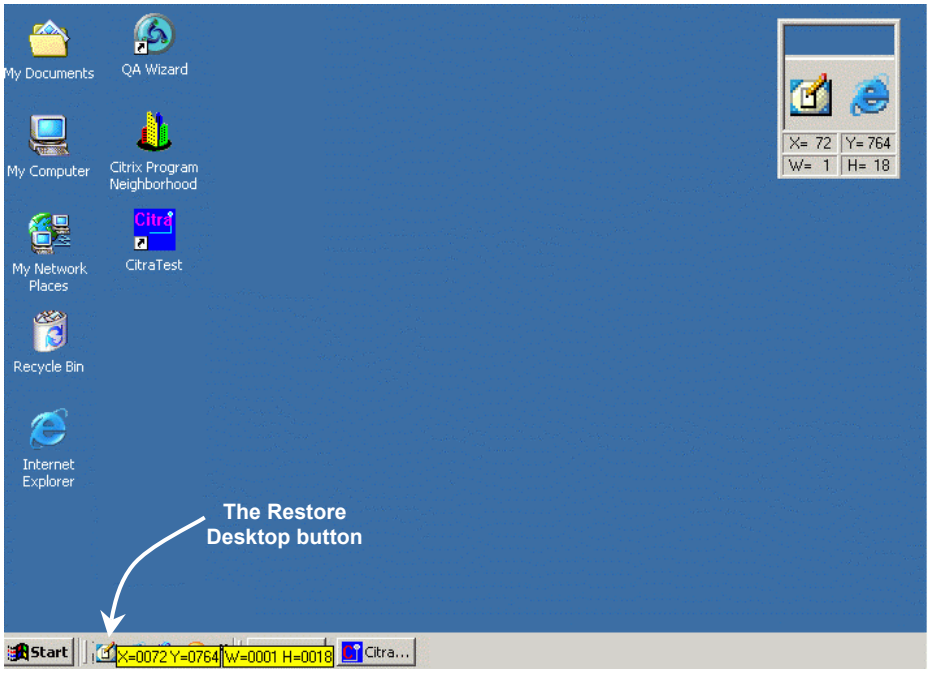

Figure 2.8: Capturing the image of the Restore Desktop button

12. While the image is being captured, a small window will appear indicating the X and Y coordinates defining the current position of the image, the width of the image, and its height (see Figure 2.8).
13. Once you release the left mouse button, you will return to Figure 2, which will now display the captured image (see Figure 2.9).

Note:

Remember the following while capturing an image:

- It is recommended that you keep the images **small**. Larger the image, longer the time taken for script execution.
- It is not always necessary for you to capture an entire image. Sometimes, capturing a small portion of the image will suffice. This is more so in the case of images that need not be clicked on. Therefore, before attempting to capture an image, ascertain its purpose and then proceed.

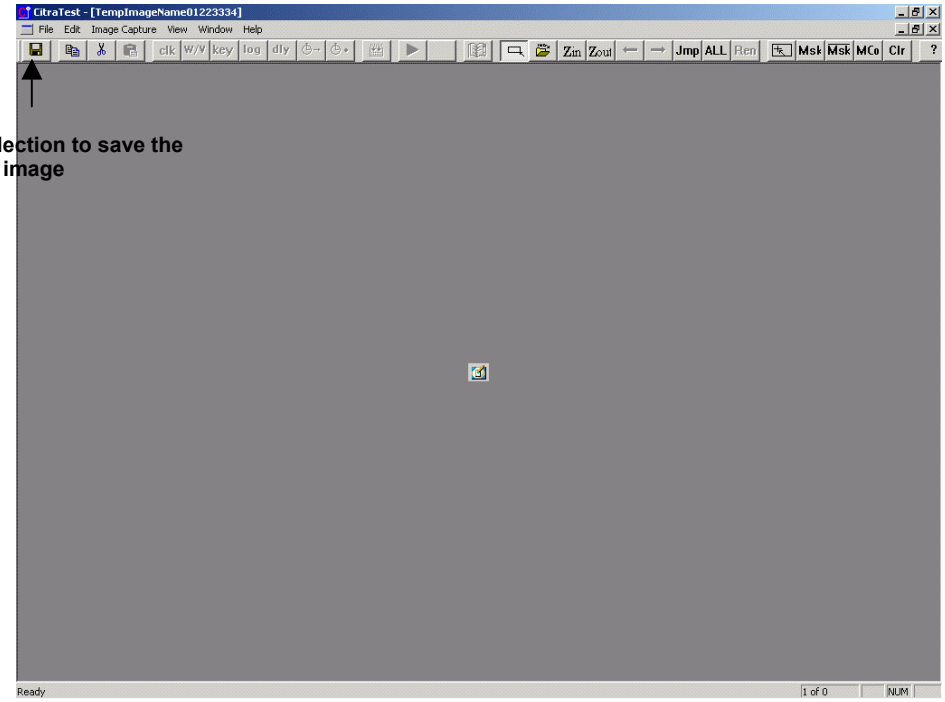

Figure 2.9: The captured Restore Desktop image

14. Save the image by clicking on the **Save** button on the tool bar of Figure 2.9. As already mentioned, by default, images captured will be stored as BMPs in the **Images** directory. Hence, by default, the **Images** directory will open in the **Save As** dialog box. Specify a name for the image (in our example, this is **RestoreDesktop**) in the **Filename** text box, and click the **Save** button in Figure 2.10 to save the image to the **Images** directory.

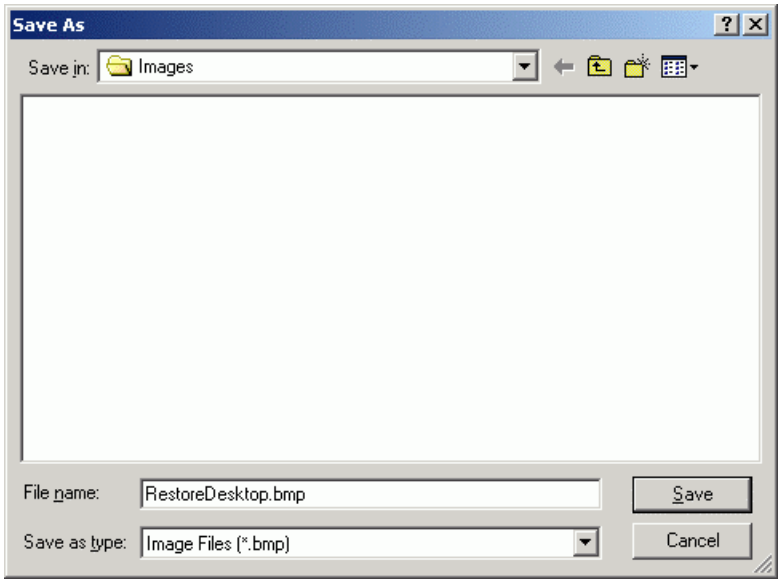

Figure 2.10: Saving the Restore Desktop image

15. Now that the image has been saved, it is now time to instruct the VB script to click on the **Restore Desktop** button. To achieve this, first minimize the image window of Figure 2.9. This will reveal the script window depicted by Figure 2.7. Now, click on the **clk** button on the tool bar of Figure 2.7. Figure 2.11 will then appear wherein you need to select the image to be clicked on from the **Image Names** list box. This list box will display all the images that are currently available in the **Images** directory. In our example, since **RestoreDesktop.bmp** is the only image currently available in that directory, only **RestoreDesktop.bmp** is displayed in the **Image Names** list box. Since this is the image to be clicked on, select it.

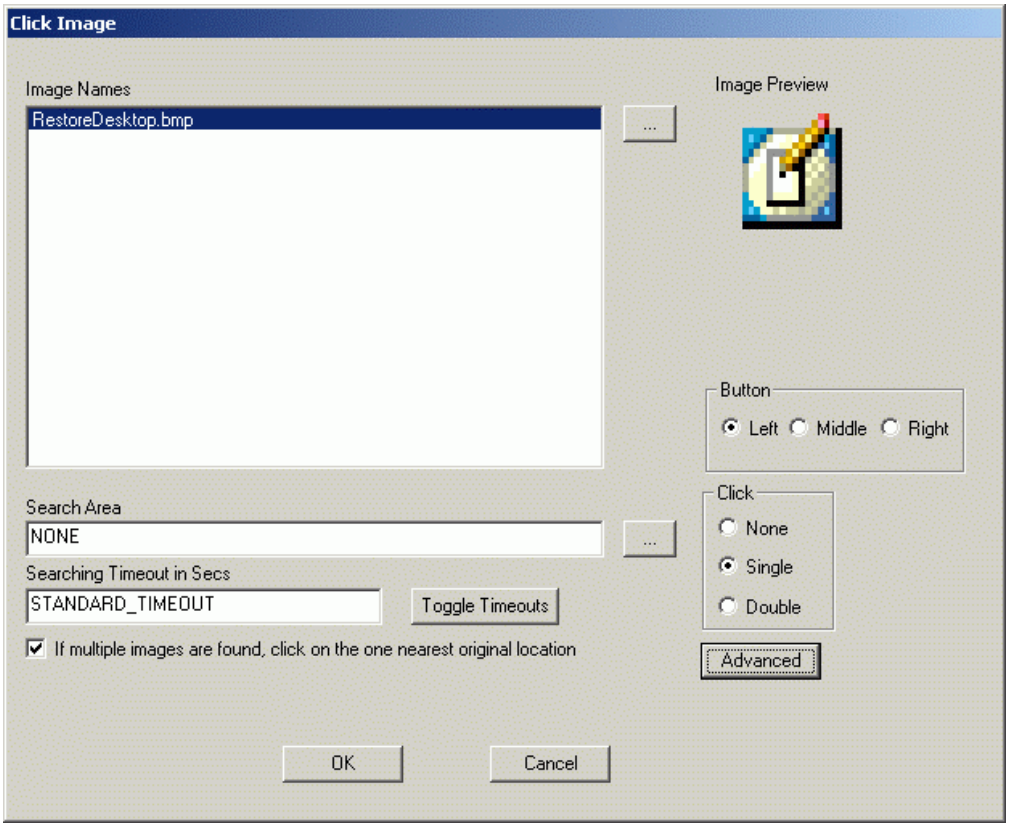

Figure 2.11: Clicking on the Restore Desktop button

16. If the chosen image has to be searched for in a search area, specify the name of the search area in the **Search Area** text box (see Figure 2.11). As no search areas have been defined yet for our example, leave it as **NONE**. By default, the script will search for the chosen image for 50 seconds, after which it will timeout. This is the **STANDARD_TIMEOUT** period for a script. If the **STANDARD_TIMEOUT** period does not apply to the selected image, you can specify a different value in the **Searching Timeout in Secs** text box. For our example, however, the **STANDARD_TIMEOUT** period holds good (see Figure 2.11).

Note:

By default, the **STANDARD_TIMEOUT** period is 50 seconds. You can change this default setting using the **Playback Options** dialog box that appears upon selecting **Playback Options** from the **Playback** menu on the script window.

17. Next, select the mouse button to be used for clicking on the selected image. The **RestoreDesktop.bmp** image in our example is to be clicked on using the default **Left** mouse button (see Figure 2.11). Therefore, select the **Left** option from the **Button** section. Similarly, since the **RestoreDesktop.bmp** image is to be clicked only once, select the **Single** option from the **Click** section (see Figure 2.11).
18. Finally, click on the **OK** button in Figure 2.11.

19. Upon clicking, CitraTest will automatically create a script corresponding to the click event that was just configured. This script will automatically appear in the script window (see Figure 2.12).

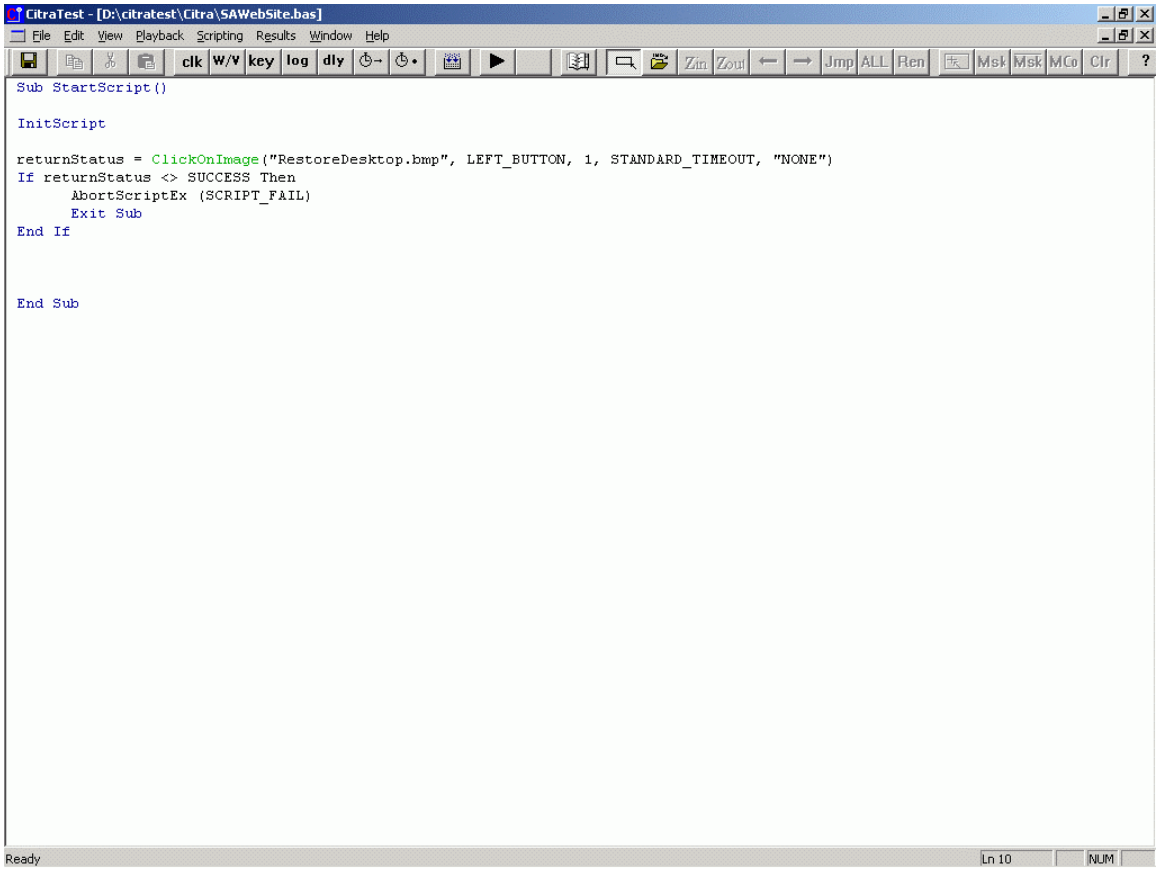

Figure 2.12: VB script for clicking on the RestoreDesktop.bmp

20. Also, note that the default termination logic (of Figure 2.6) has been appended to the script (see Figure 2.12). This means that if the click event fails, the termination script will be invoked and script execution will be immediately aborted.

21. The next step is to ensure that the desktop refreshes once the **Restore Desktop** button is clicked. Pressing the F5 key on the keyboard refreshes the desktop. To instruct the VB script to do the same, click on the **key** button on the tool bar of Figure 2.12.

22. Figure 2.13 will then appear. From the **Special Keys** list of Figure 2.13, select F5. As the **F5** key is to be "pressed", select the **Key Press** option from the **Keystroke Action for Special Keys** section. Next, click on the **Add to Script Code >>** button adjacent to the **Special Keys** list. When this is done, the script code corresponding to the "F5 key press" will appear in the **Keystroke Script Code** list (see Figure 2.14). Finally, click on the **OK** button in Figure 2.14 to add the script code to the script window (see Figure 2.15).

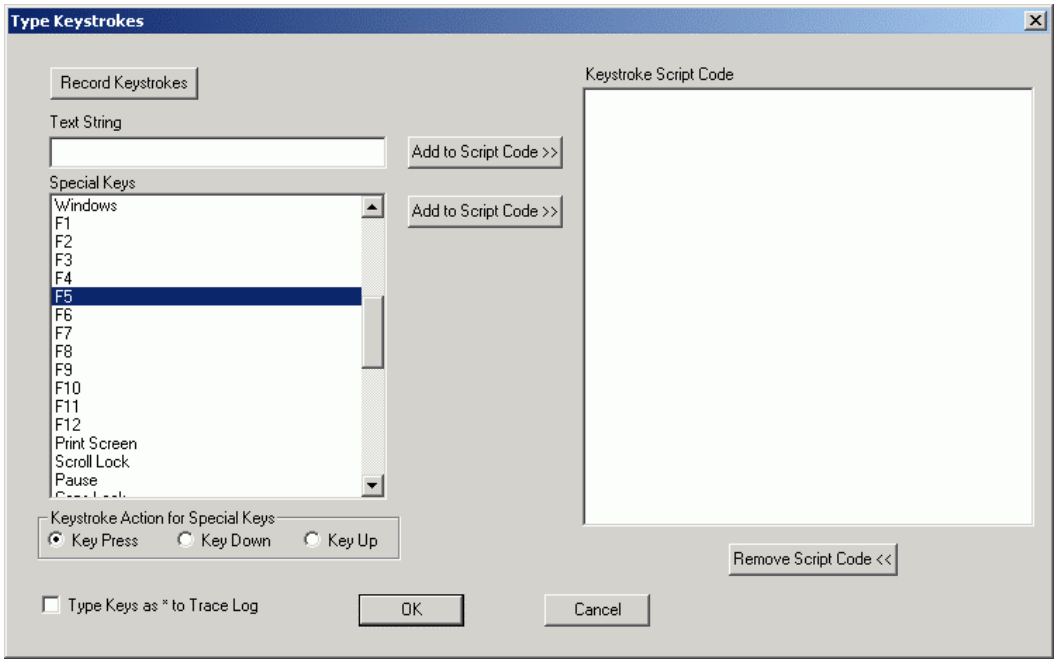

Figure 2.13: Selecting the F5 key

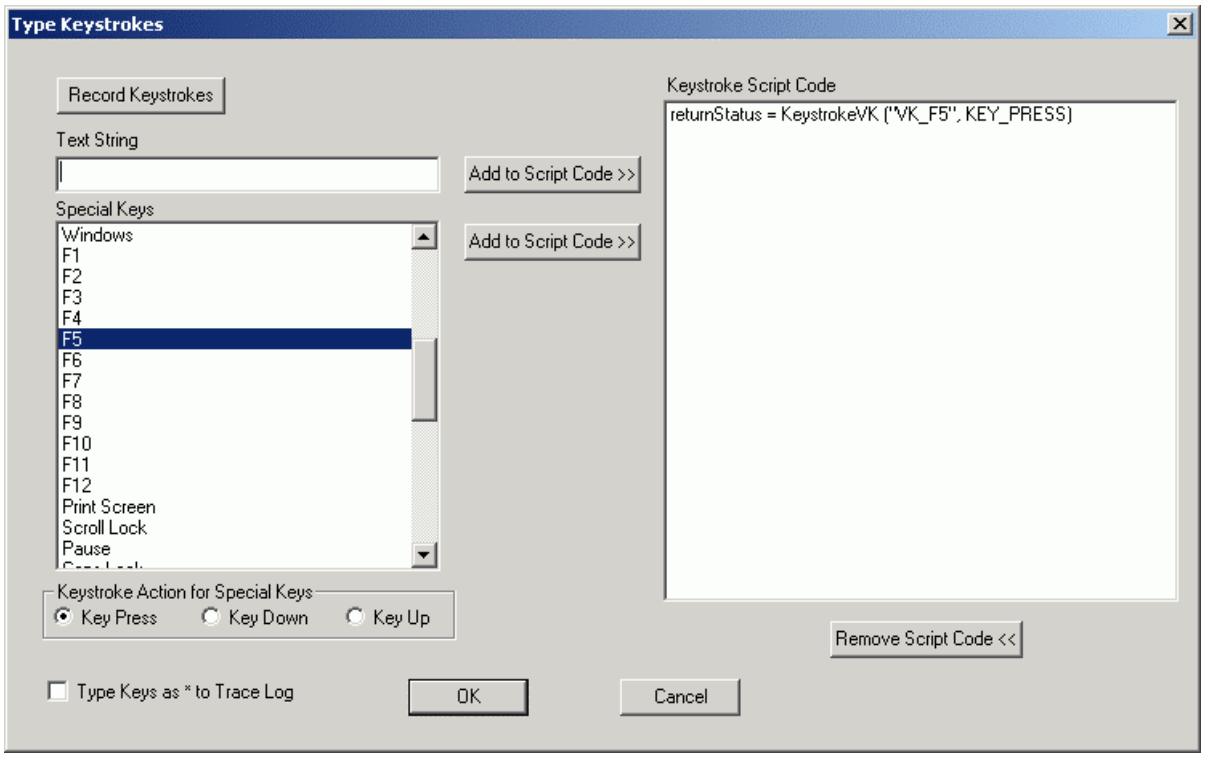

Figure 2.14: Adding the F5 key to the script

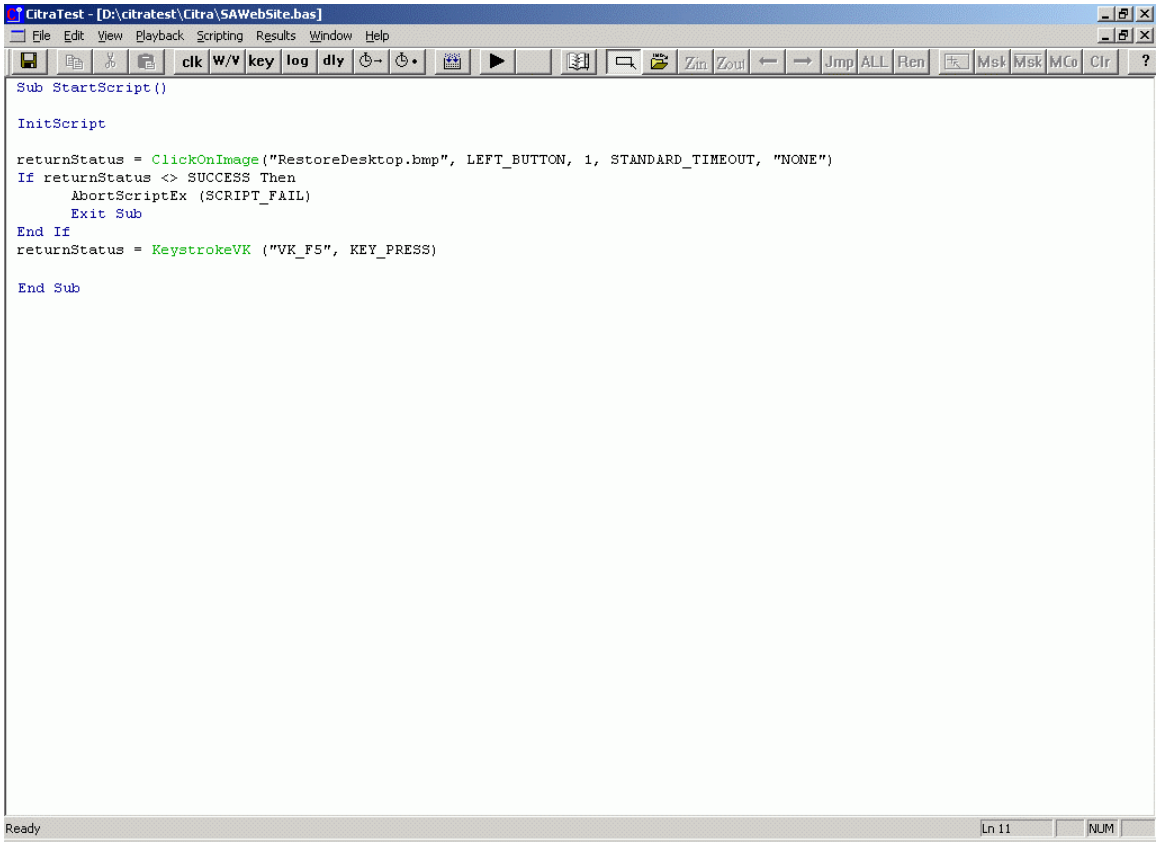

Figure 2.15: Script for pressing the F5 key

23. The script will now ensure that the Windows desktop is restored and refreshed. The step to be handled next by the script is, double-clicking on the **Internet Explorer** icon on the desktop to open the Internet Explorer. To achieve this, an image of the **Internet Explorer** icon needs to be first captured, and then a click event needs to be associated with it.

24. To capture the **Internet Explorer** icon's image, first, press the F11 key on the keyboard to enable image capturing. Capture the required icon using the same procedure explained earlier (see Figure 2.16).

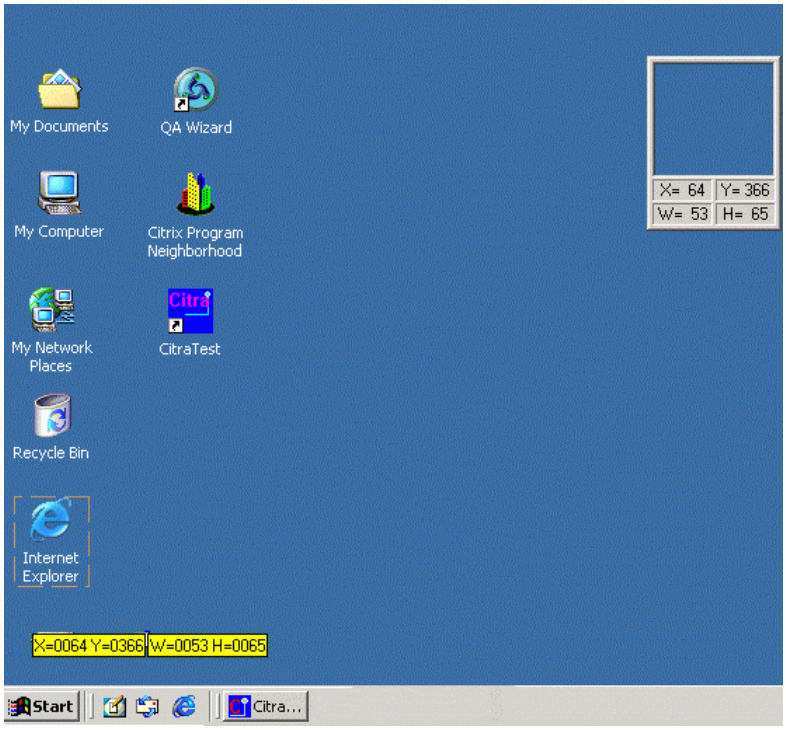

Figure 2.16: Capturing the Internet Explorer Icon

25. Upon releasing the left mouse button, Figure 2.17 will appear using which the captured image is to be saved to the **Images** directory. From Figure 2.17, it is evident that the **Internet Explorer** icon in our example has been saved as **IE.bmp** (see Figure 2.17).

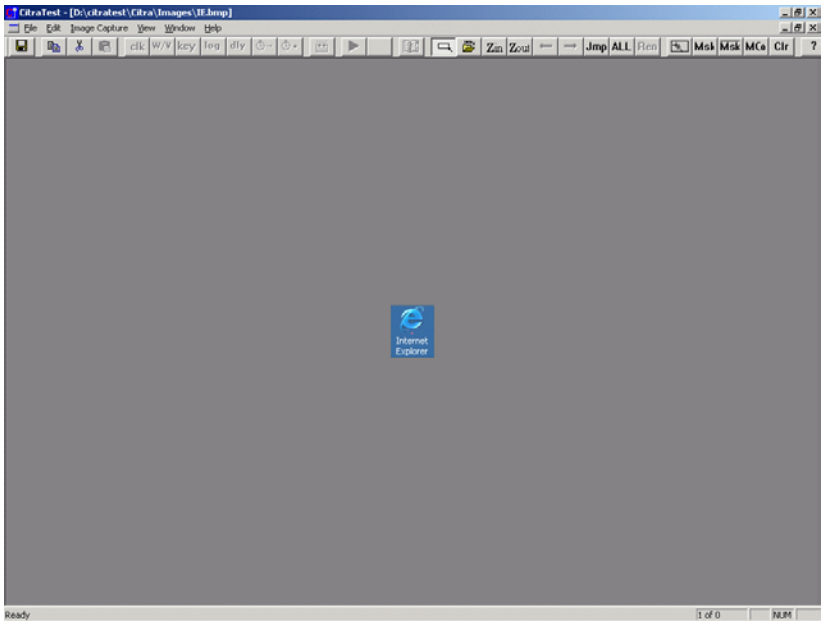

Figure 2.17: The saved Internet Explorer icon

26. Every image that is captured using CitraTest will consist of a default **Click** spot. If a click event is associated with an image, then, during playback, the VB script will attempt to click only on the image's **Click** spot. If this spot is not clearly defined, the VB script will be unable to click on the corresponding image, and eventually, the script will fail. Therefore, if you even slightly suspect the correctness of the default **Click** spot, then it is recommended that you change it immediately. The default **Click** spot associated with the **IE.bmp** in our example, has been indicated by Figure 2.18.

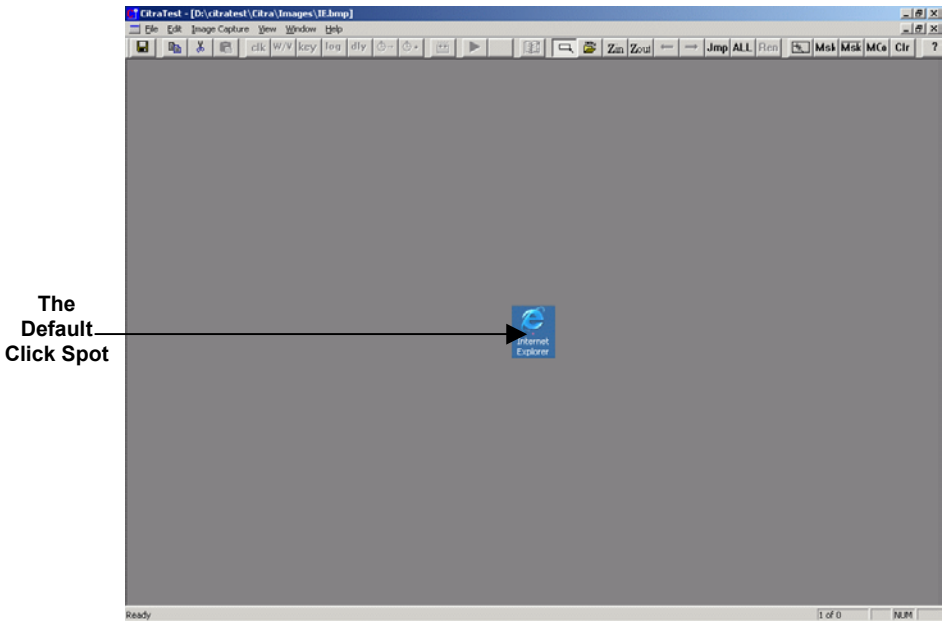

Figure 2.18: The default click on spot of IE.bmp

27. In Figure 2.18, the default **Click** spot appears as a small, red dot just above the phrase, **Internet Explorer**. Say that this spot needs to be redefined. To do so, first, click on the  button on the tool bar of Figure 2.18.

28. Then, bring your cursor down to the **IE.bmp** image and click on the spot that needs to be set as the new **Click** spot. Figure 2.19 reveals the new click spot for the **IE.bmp** in our example.

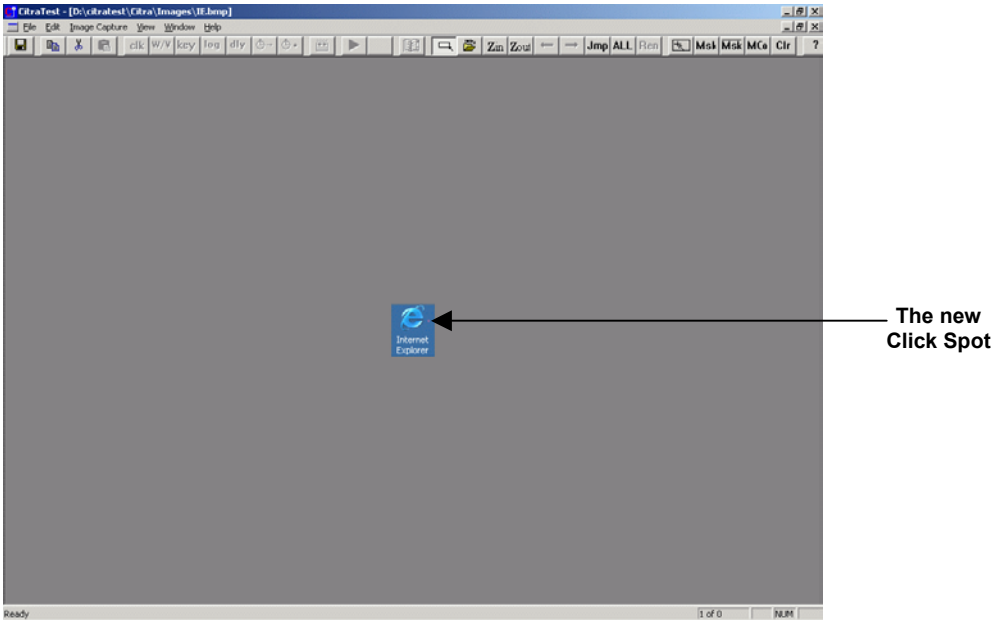

Figure 2.19: Defining a new click spot for IE.bmp

29. Next, associate a click event with the **IE.bmp**. To do so, first, minimize Figure 2.17 to open Figure 2.15. Then, click on the **clk** button on the tool bar of Figure 2.15. From the **Image Names** list box of Figure 2.20 that comes up, select **IE.bmp**, and then select the **Left** option from the **Button** section. Since the **IE.bmp** has to be double-clicked, choose the **Double** option from the **Click** section, and finally, click the **OK** button to generate the corresponding VB script.

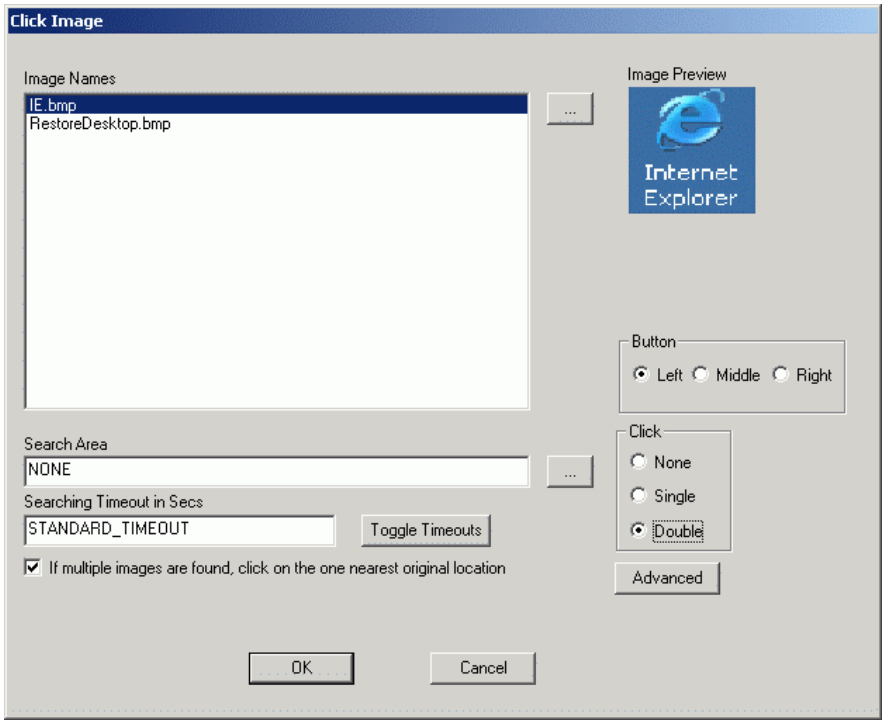

Figure 2.20: Clicking on the IE.bmp

30. Figure 2.21 will then appear bearing the VB script that was automatically generated for the above-mentioned condition.

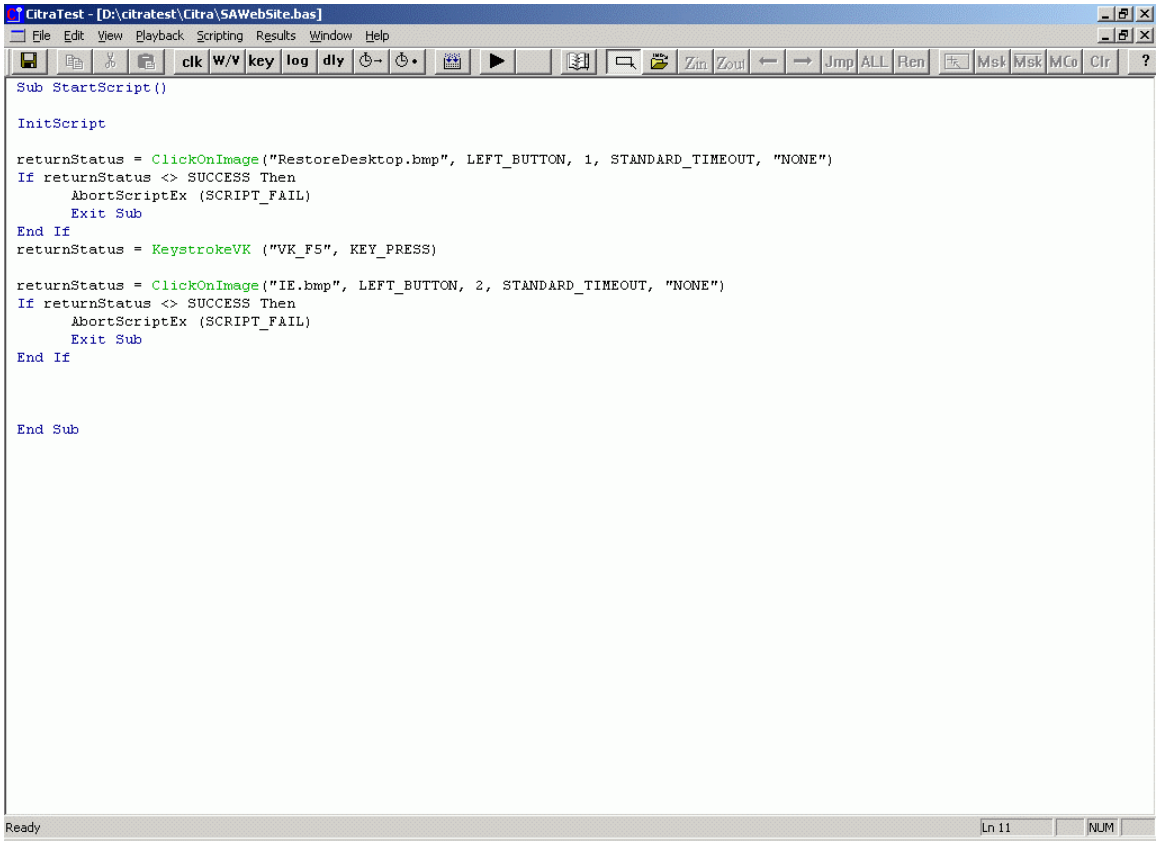

Figure 2.21: Script for double-clicking on the IE.bmp

31. Here again, note that the default termination logic has been appended to the script.

32. The complete script now takes care of restoring the desktop, refreshing it, and double-clicking on the Internet Explorer icon on the desktop. When the IE icon is double-clicked, naturally, the Internet Explorer window will open. Once the IE window opens, the URL of the **Singapore Airlines** web site should be entered in the **Address** box of the window, so that the web site is accessed. The URL is: http://www.singaporeair.com/saa/app/saa.

33. The first step towards typing the URL in the **Address** box of the Internet Explorer, is to click on the **Address** box. To achieve this, first, capture the image of the **Address** box by pressing F11 to enable image capturing, and proceeding in the manner depicted by Figure 2.22 below.

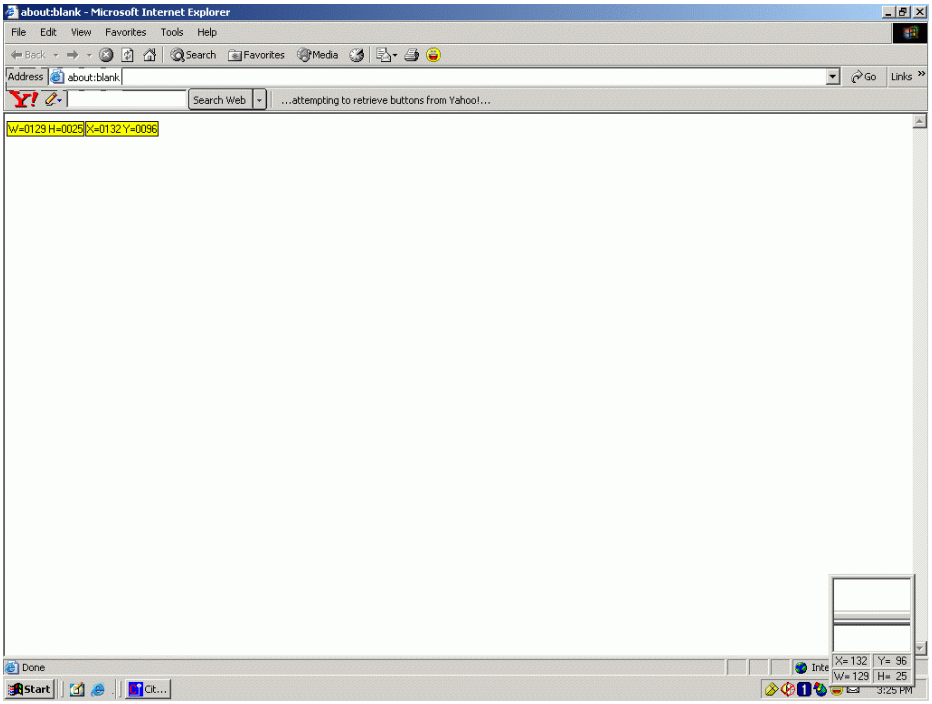

Figure 2.22: Capturing the image of the Address box

34. Figure 2.23 reveals that the image has been captured and has been saved as **addressbar.bmp**.

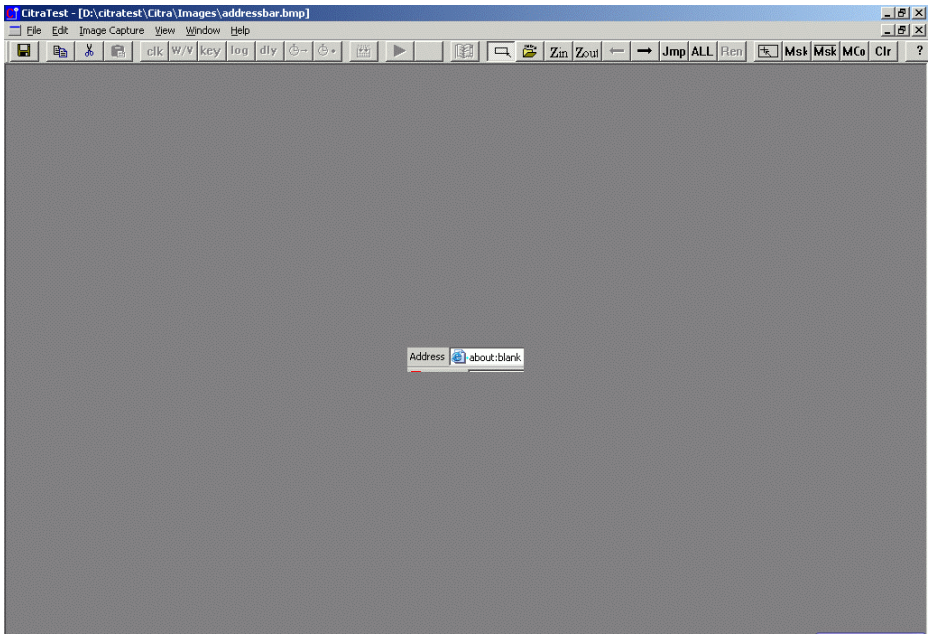

Figure 2.23: The captured image has been saved as addressbar.bmp

35. Typically, the **Address** box will contain the default URL to connect to. As soon as the Internet Explorer opens, it will try and connect to the default URL only. This default setting is system-specific, i.e., it will vary from one system to another. In our example,

the default URL is *about:blank* (see Figure 2.23). If the same script is executed on another system, it will look for *about:blank* in the **Address** box of the IE in that system also. If the script does not find the same entry, it will fail. To avoid this confusion, it is imperative that the script be built in such a way that the default contents of the **Address** box are ignored.

36. To achieve this, the text area of the **Address** box needs to be masked. For that, first, click on the **Msk** button (fifth from the right) on the tool bar of Figure 2.24. Then, move to the image below and mark the text box area that is to be masked (see Figure 2.24). Once the text area is masked, the color of the whole image will change as indicated by Figure 2.24. This color-change is not permanent. The original color of the image and its new color will flash alternately. Save the image once again.

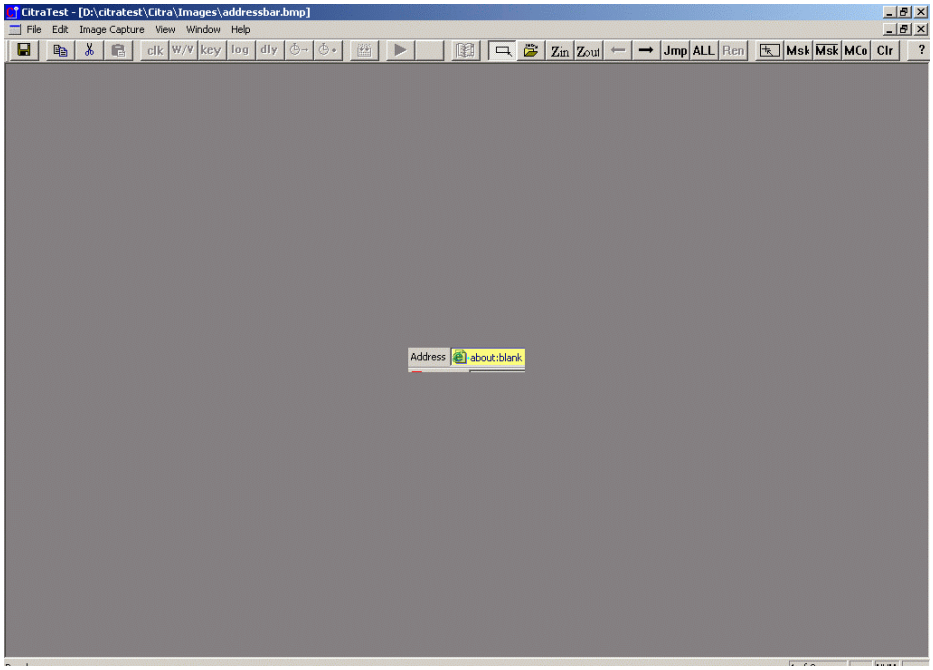

Figure 2.24: Masking the text area of the Address box

37. Next, associate a click event with the **addressbar.bmp** image. Once again, minimize the image window (see Figure 2.24) to open the script window (Figure 2.21), click on the **clk** button on the tool bar of the script window, select **addressbar.bmp** from the **Image Names** list of Figure 2.25, select the **Left** option from the **Button** section, and the **Single** option from the **Click** section. Finally, click on the **OK** button.

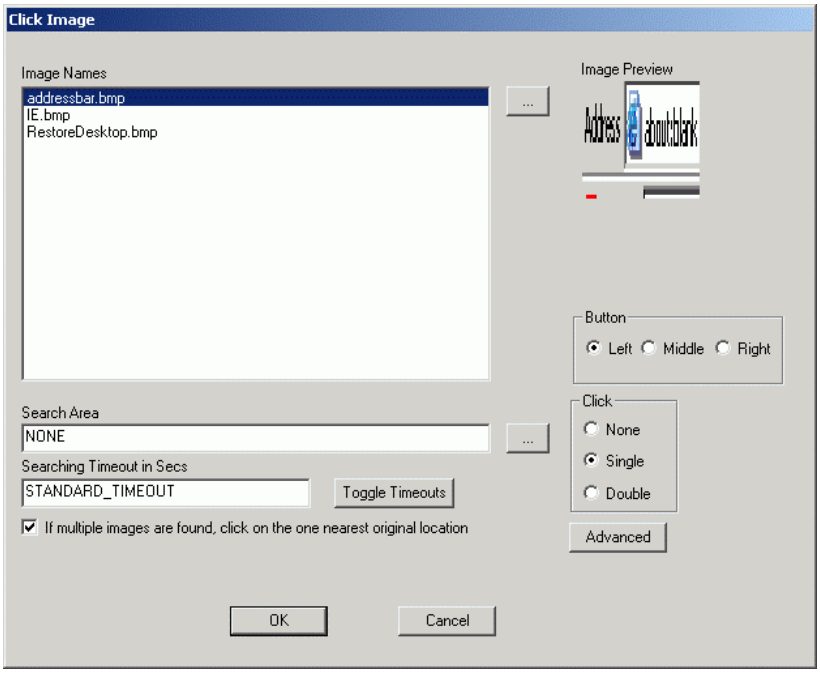

Figure 2.25: Clicking on the Address box

38. A script to the above effect and the default termination logic will then appear in the script window. Now, proceed to type the URL of web site in the **Address** box. To do so, first, click on the **key** button on the tool bar of Figure 2.26.

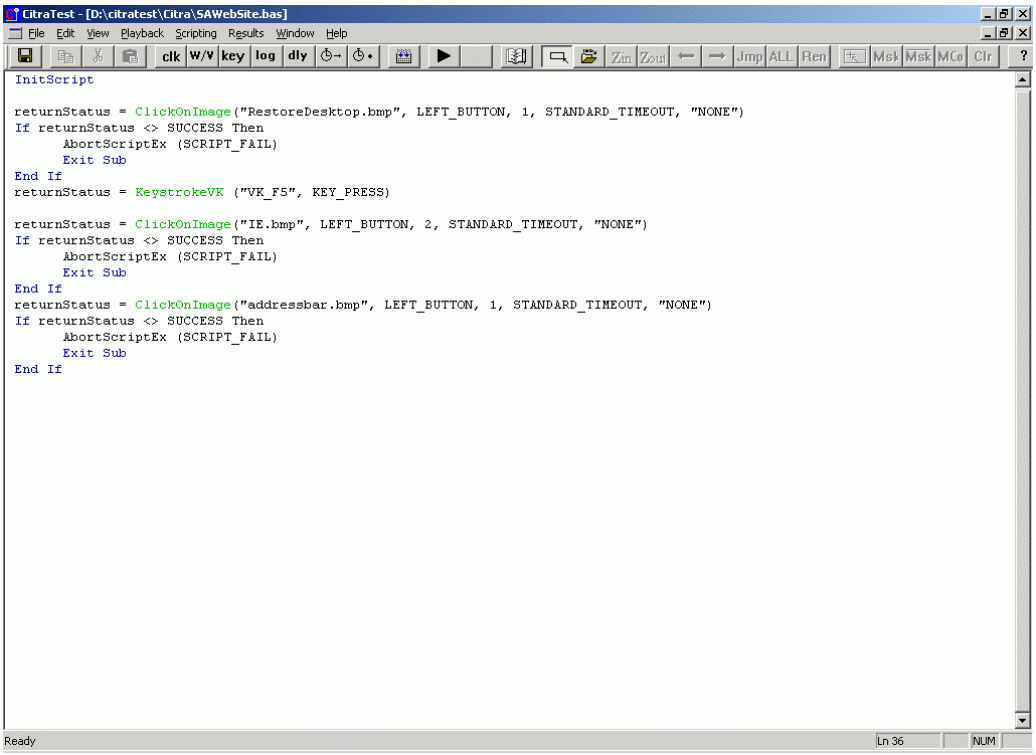

Figure 2.26: Script for clicking on the Address box

39. In the **Text String** text box of Figure 2.27 that appears, specify the complete URL of the **Singapore Airlines** site. This would be, http://www.singaporeair.com/saa/app/saa. Then, click on the **Add to Script Code >>** button corresponding to the **Text String** text box. This will generate a script code to the above effect and display it in the **Keystroke Script Code** box (see Figure 2.28).

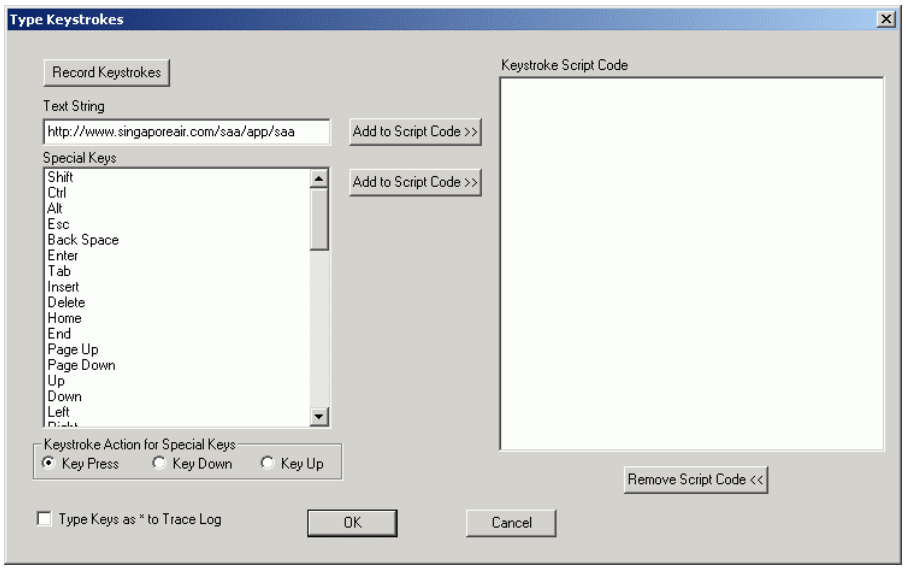

Figure 2.27: Typing the web site's URL

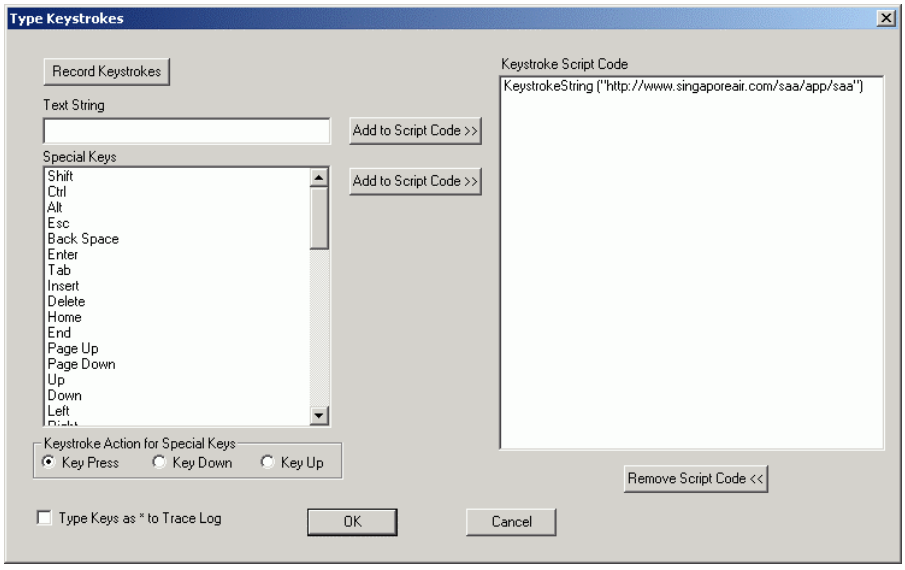

Figure 2.28: Generating the script code for typing the URL

40. Once the URL is typed, the **Enter** key on the keyboard should be pressed to initiate the web site connection. In order to make sure that the script does the same, select the **Enter** option from the **Special Keys** list box and click on the **Add to Script Code >>** button adjacent to it (see Figure 2.29). The corresponding script code will then be displayed in the **Keystroke Script Code** box (see Figure 2.30).

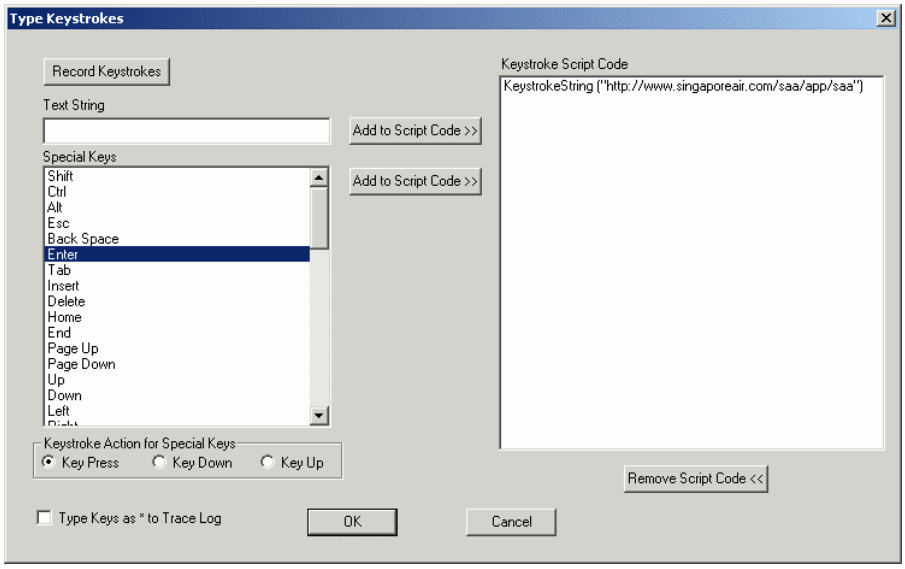

Figure 2.29: Selecting the Enter key

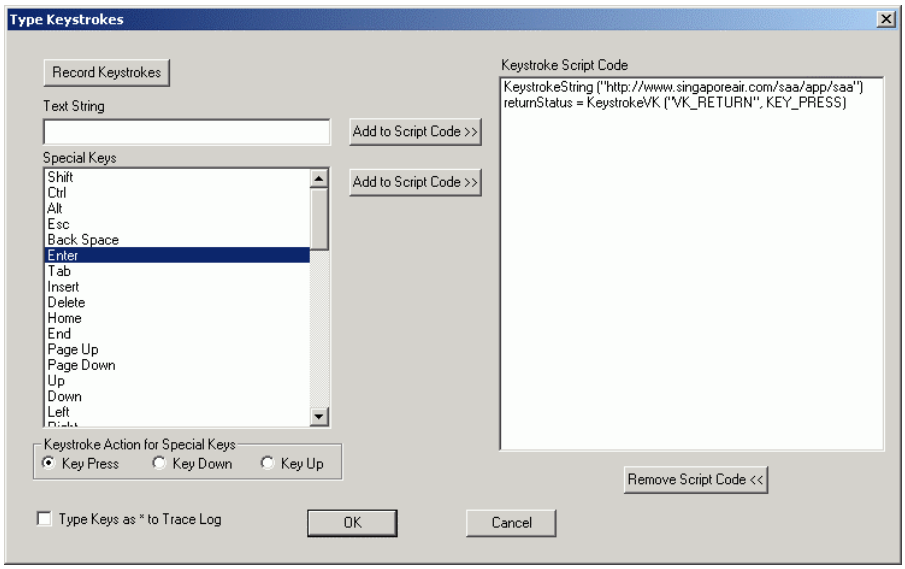

Figure 2.30: Script code for the Enter key press

41. Finally, click on the **OK** button in Figure 2.30.
42. The **Keystroke Script Codes** that were generated earlier (see Figure 2.30) will then be appended to the script window of Figure 2.31.

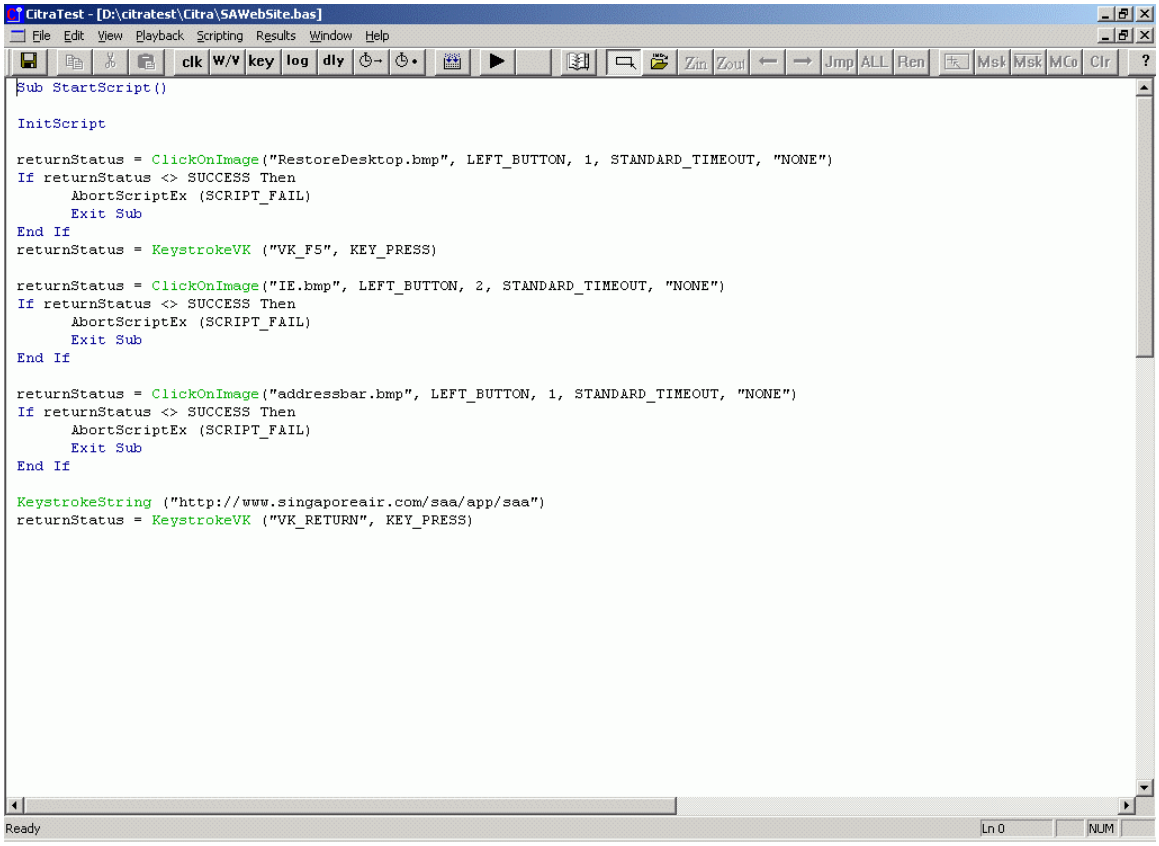

Figure 2.31: Script for typing the URL and pressing the Enter key

43. Once the URL is typed, the Singapore Airlines home page will be downloaded. The next step therefore, is to ensure that the page has downloaded successfully. Here, two images indicate the successful page download.

    a. The **Singapore Airlines** logo

    b. The **Done** message on the status bar

    As we need to keep the images small, capture a small part of the **Singapore Airlines** logo as depicted by Figure 2.32.

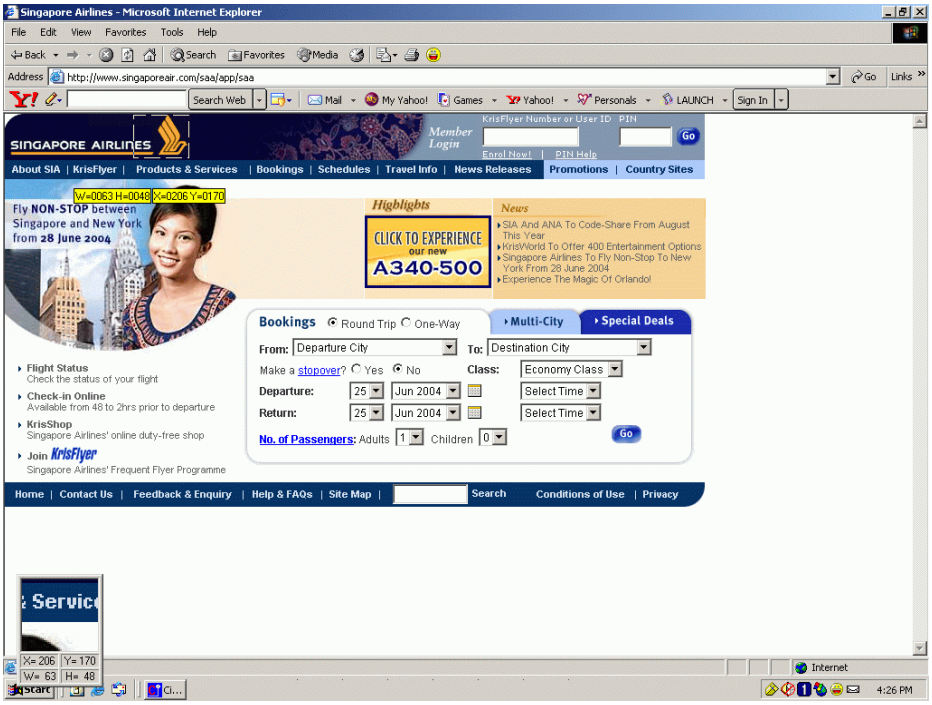

Figure 2.32: Capturing the image of the Singapore Airlines logo

44. The image has been saved as **SAHome.bmp** (see Figure 2.33).

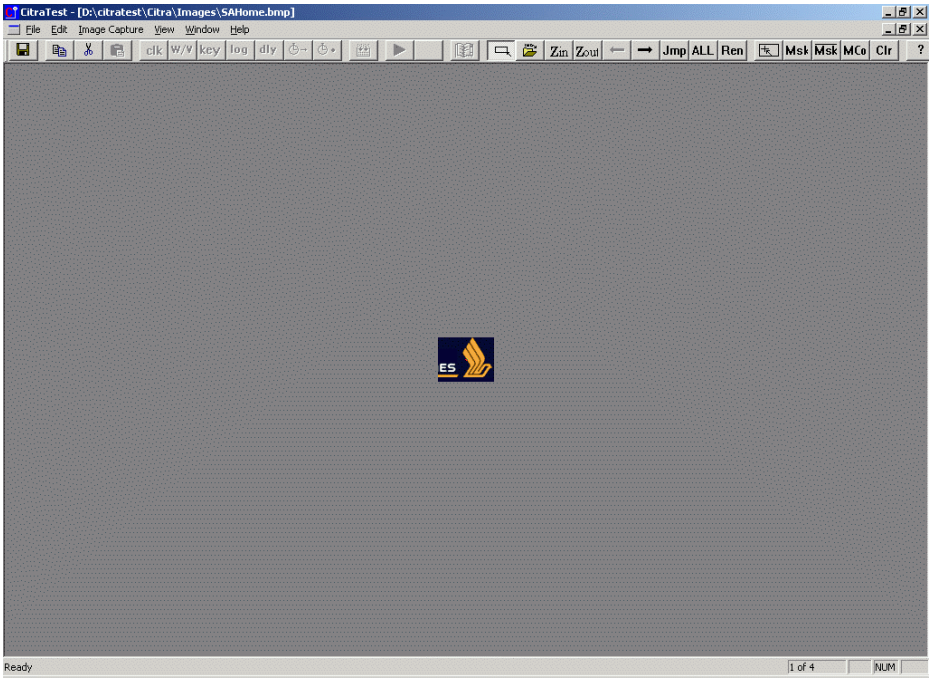

Figure 2.33: Saving the Singapore Airlines logo as SAHome.bmp

45. Then, capture the **Done** message on the status bar by following the procedure explained earlier. Save the new image as **DoneSAHome.bmp**.

46. Once both the images are ready, you need to ensure that the script waits for the appearance of the logo and the **Done** message. For that, first, switch to the script window and click on the **W / V** button on its tool bar. Then, select the **DoneSAHome** and **SAHome** images from the **Image Names** list (see Figure 2.34), choose the **Wait For ALL Images** option, and click the **OK** button in Figure 2.34.

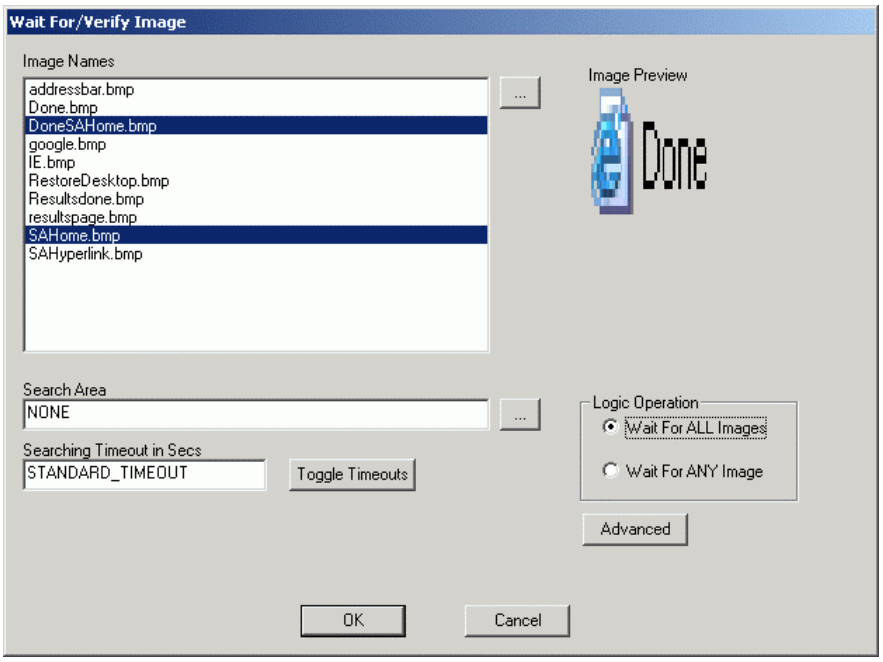

Figure 2.34: Generating a "wait for" script for the SAHome and DoneSAHome images

47. The "wait for" script will then appear in the script window (see Figure 2.35).

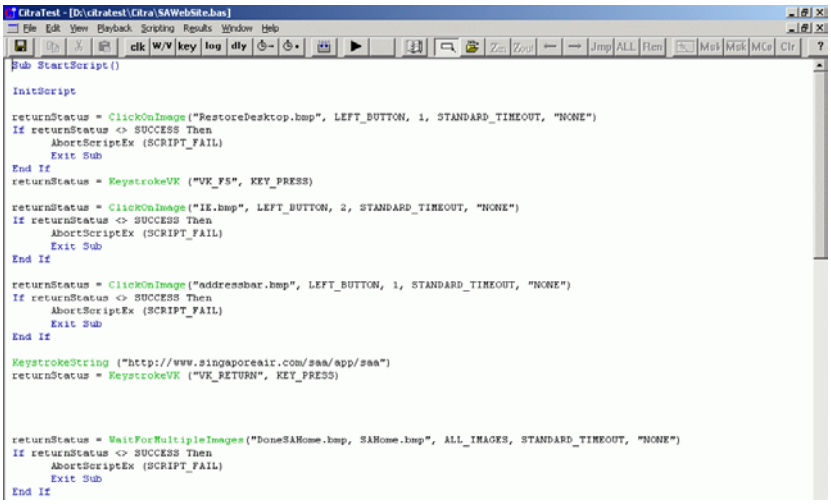

Figure 2.35: Script that will wait for the display of the Singapore Airlines logo and Done message

48. Next, let us try to build into the script, the capability to track the time taken by the **Singapore Airlines** home page to download. The **Timer** component of CitraTest facilitates this tracking. To insert a timer component, first, place the cursor just above the "wait for"

script in the script window (see Figure 2.35). Then, click on the button on the tool bar of Figure 2.35. Figure 2.36 appears, prompting you to provide a name for the timer that is being inserted. Let us call the timer in our example *SAHome*. Clicking on the **OK** button in Figure 2.36 will open the script window, which will now display the code, *StartTimer ("SAHome")*, just above the "wait for" script (see Figure 2.37).

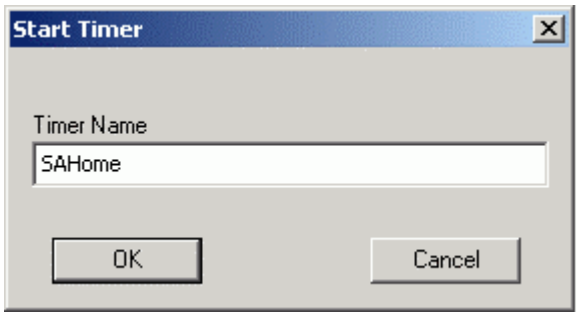

Figure 2.36: Inserting the *SAHome* timer for tracking the download time of the Singapore Airlines home page

49. Stop the timer after the "wait for" script (see Figure 2.37) by clicking on the button. This will include a *StopTimer* code for the new timer (see Figure 2.37).

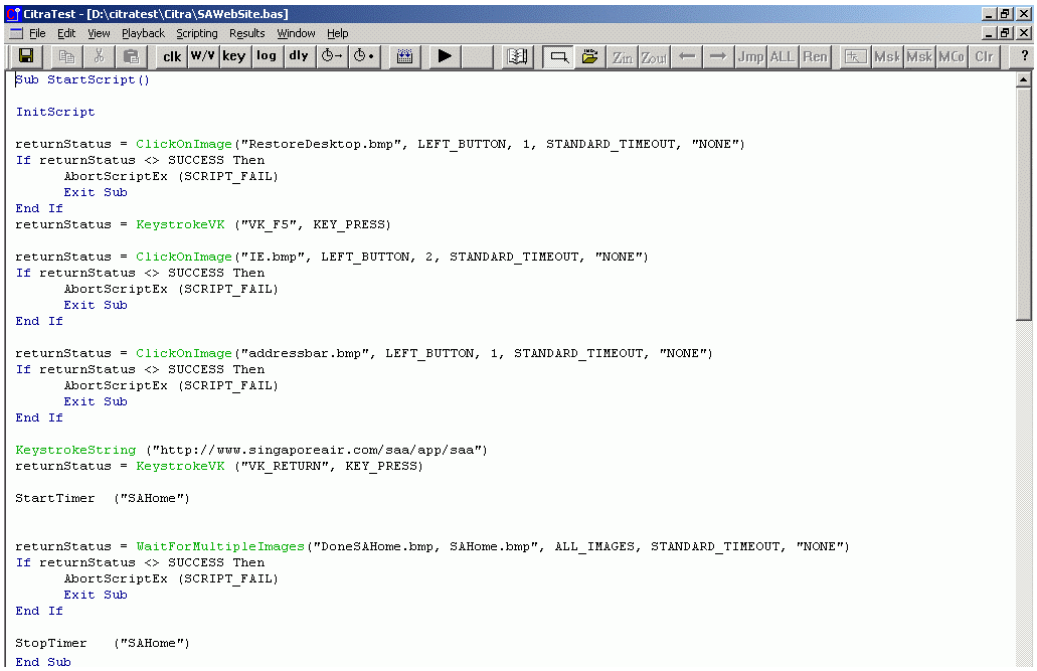

Figure 2.37: Script tracking the time taken by the Singapore Airlines page for downloading

50. The **Timer** component then calculates the difference between the start and stop times, and returns the time that the **Singapore Airlines Home** page actually took to download. This becomes the response time of that page. The timers inserted in the script will appear as "descriptors" in the eG suite, and will report the availability and responsiveness of the pages they track; in this case, the name of the descriptor will be *SAHome*, and it will report the availability and response time of the **Singapore Airlines Home** web page.

51. The script should now try to access the page which provides the Flight status. For that, the script needs to move the mouse pointer over the **Schedules** link on the panel at the top of the **Singapore Airlines** page, and then, click on the **Flight Status** option in its drop-down menu. To achieve this, first, capture the image of the **Schedules** link as depicted by Figure 2.38.

Figure 2.38: Capturing the image of the Schedules link

52. This image has been saved as **Schedules.bmp** (see Figure 2.39).

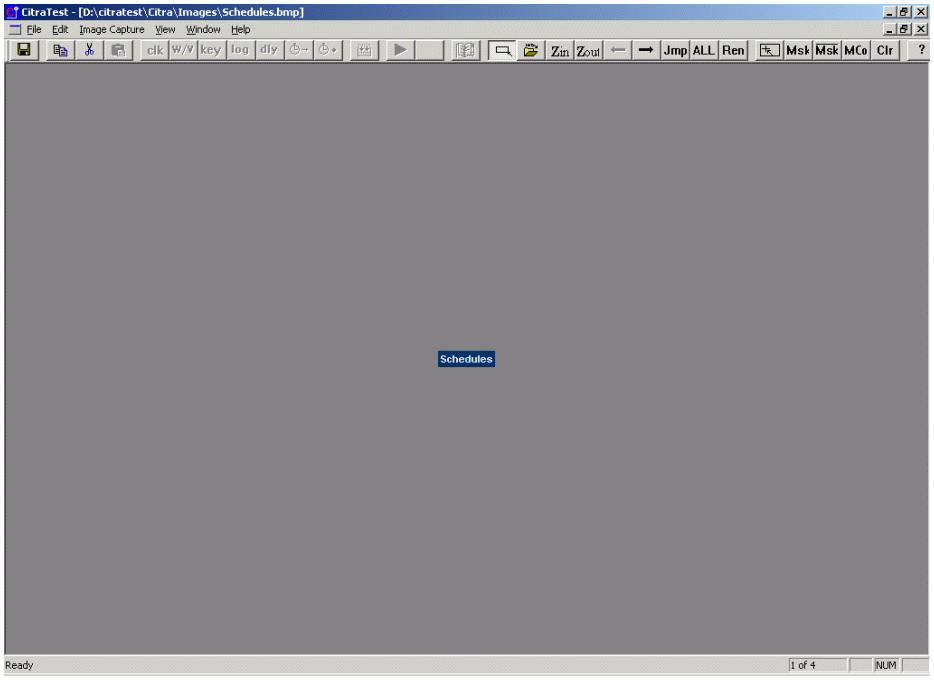

Figure 2.39: Saving the Schedules images as Schedules.bmp

53. To move the mouse pointer over the **Schedules** link, first, click on the **clk** button on the tool bar of the script window. Select the **Schedules** image from the **Image Names** list of Figure 2.40, and select the **None** option from the **Click** section. Finally, click on the **OK** button in Figure 2.40.

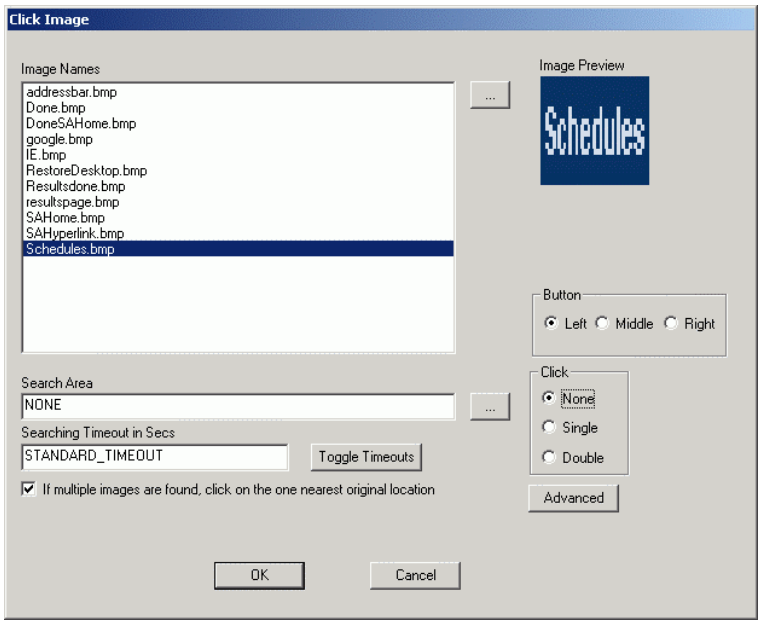

Figure 2.40: Generating the script for moving the mouse pointer over the Schedules link

54. The generated script will be appended to the script window (see Figure 2.41).

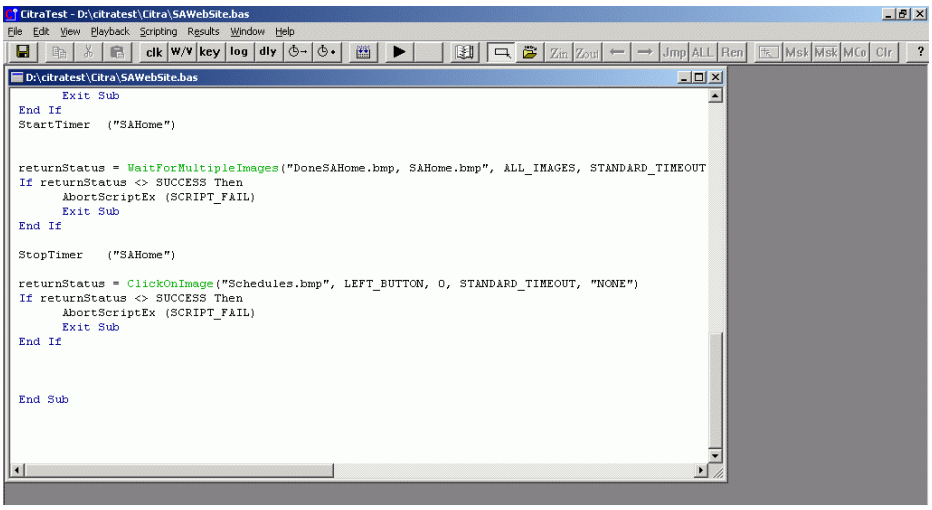

Figure 2.41: Script for for moving the mouse pointer over the Schedules link

55. Next, capture the image of the **Flight Status** option in the **Schedules** menu (see Figure 2.42).

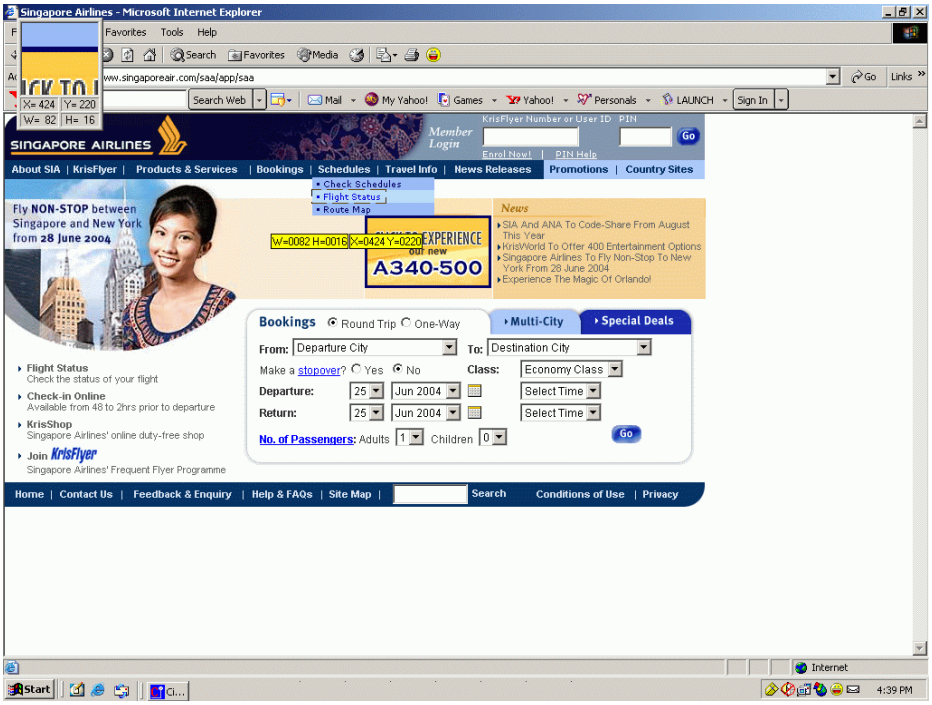

Figure 2.42: Capturing the Flight Status image

56. Name the file as **FlightStatus.bmp** (see Figure 2.43), and then, associate a click event with it as depicted by Figure 2.44.

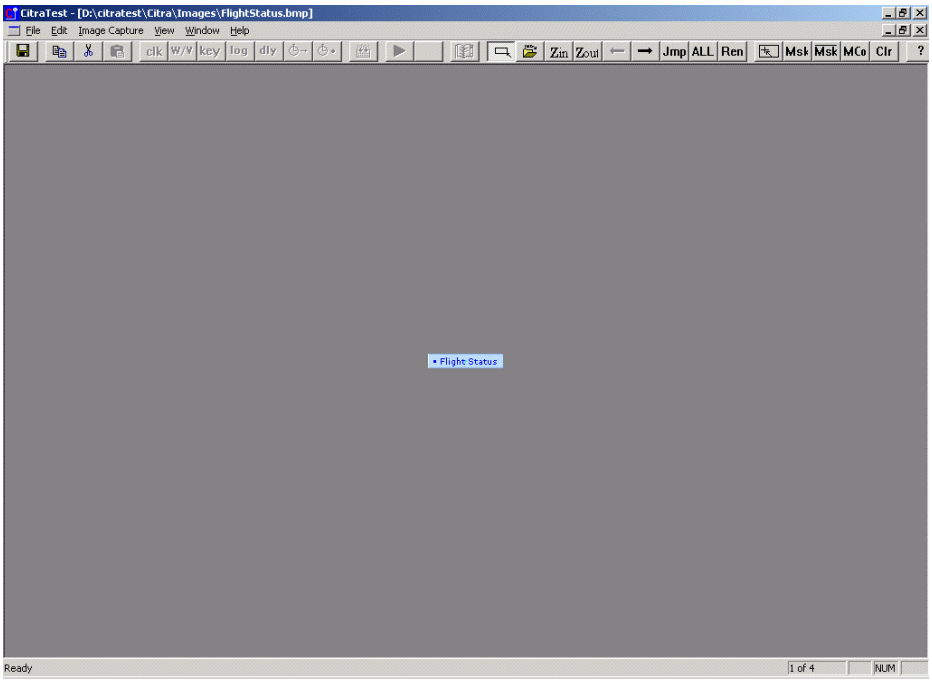

Figure 2.43: Saving the Flight Status image

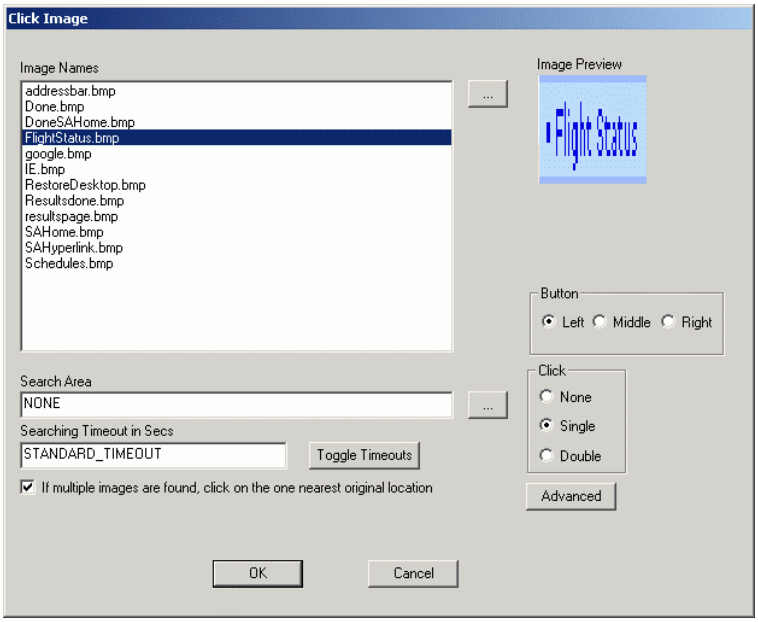

Figure 2.44: Associating a Single click with the Flight Status image

57. The script that handles the afore-mentioned will then appear (see Figure 2.45) in the script window.

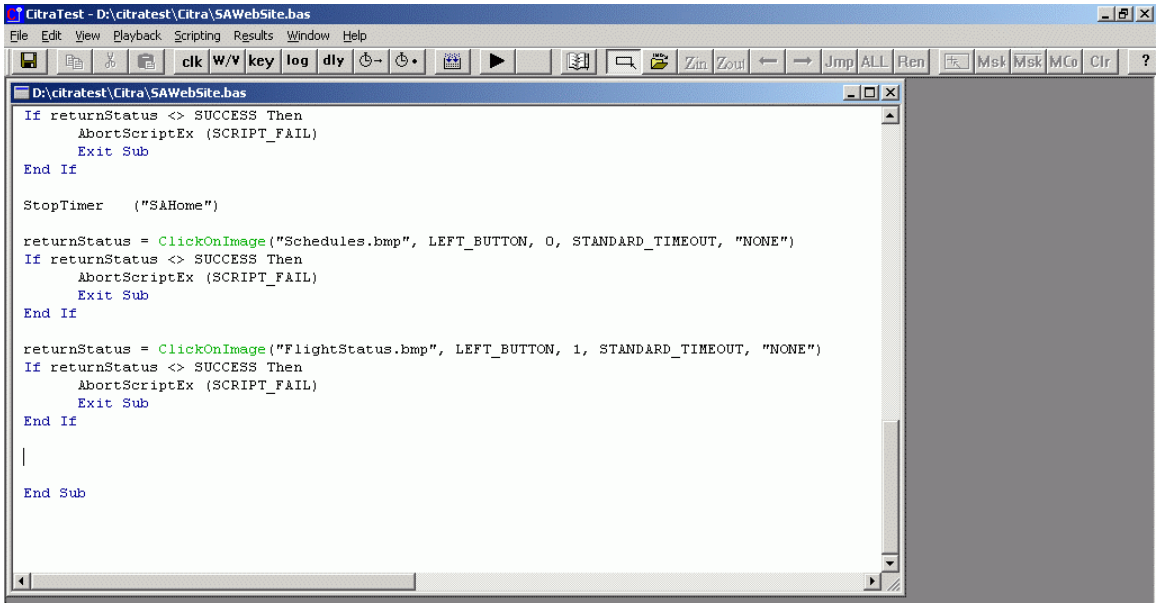

Figure 2.45: Script for clicking on the FlightStatus.bmp image

58. Upon clicking the **Flight Status** option, the **Flight Status** page will appear. Using this page, the status of a particular flight will have to be retrieved. To check whether the **Flight Status** page has completely downloaded, let us use the page title – **Flight Status** – as an indicator. The **Done** image that appears on the status bar of the IE window can also be used as an indicator. First, capture the image of a small part of the page title as depicted by Figure 2.46 below.

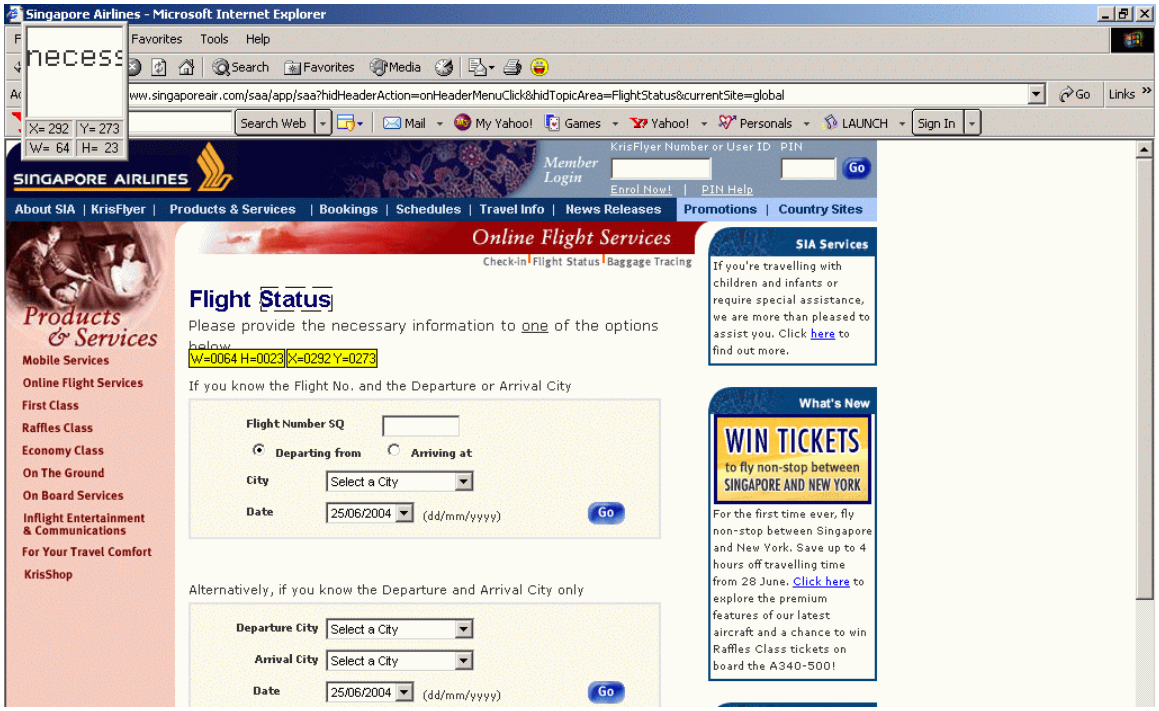

Figure 2.46: Capturing the image of the page title – Flight Status

59. Save the image as **Status.bmp**. Instead of capturing the **Done** image again, let us use one of the existing **Done** images. Now, instruct the script to wait for both these images, by first, clicking on the **W / V** button on the script window's tool bar. Then, select the **Status** image and any of the **Done** images, and click on the **Wait For ALL Images** option. Finally, click the **OK** button.

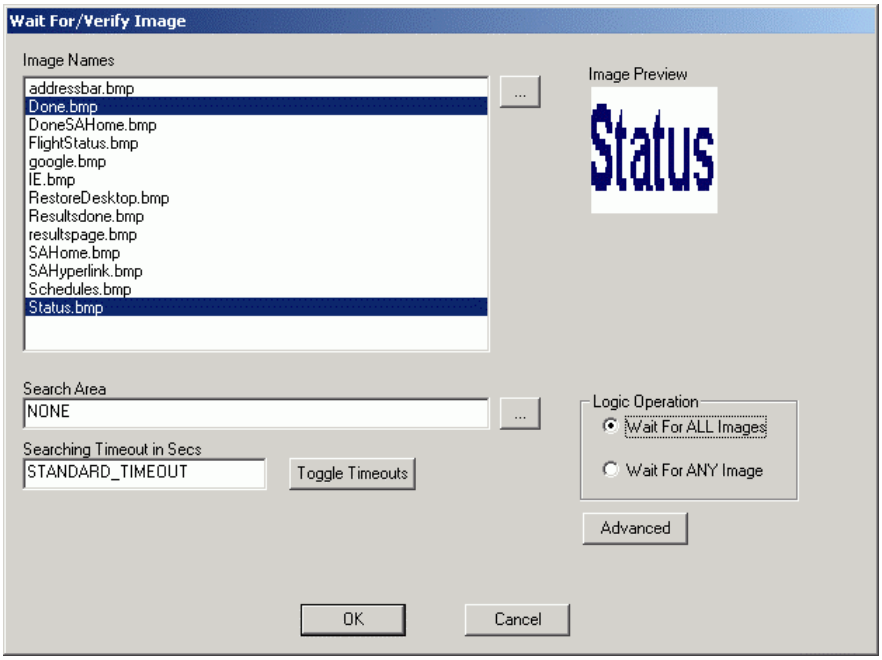

Figure 2.47: Waiting for the Done and Status images

60. Note that the script window of Figure 2.48 reveals the code for waiting for the **Done** and **Status** images.

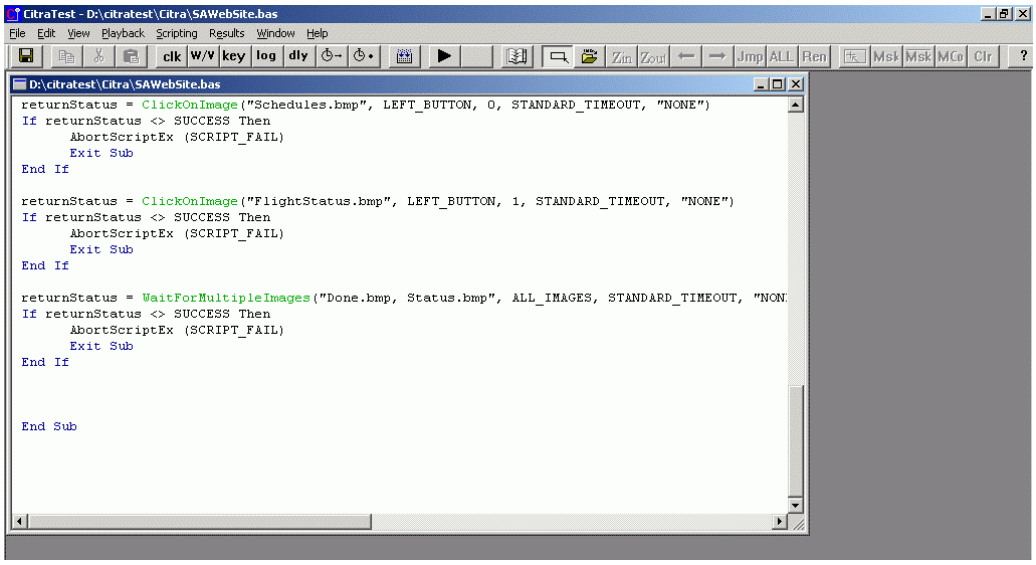

Figure 2.48: Script for waiting for the Done and Status images

61. As before, track the time taken by the **Flight Status** page to download by starting and stopping a timer named *FlightStatusPage* (see Figure 2.49).

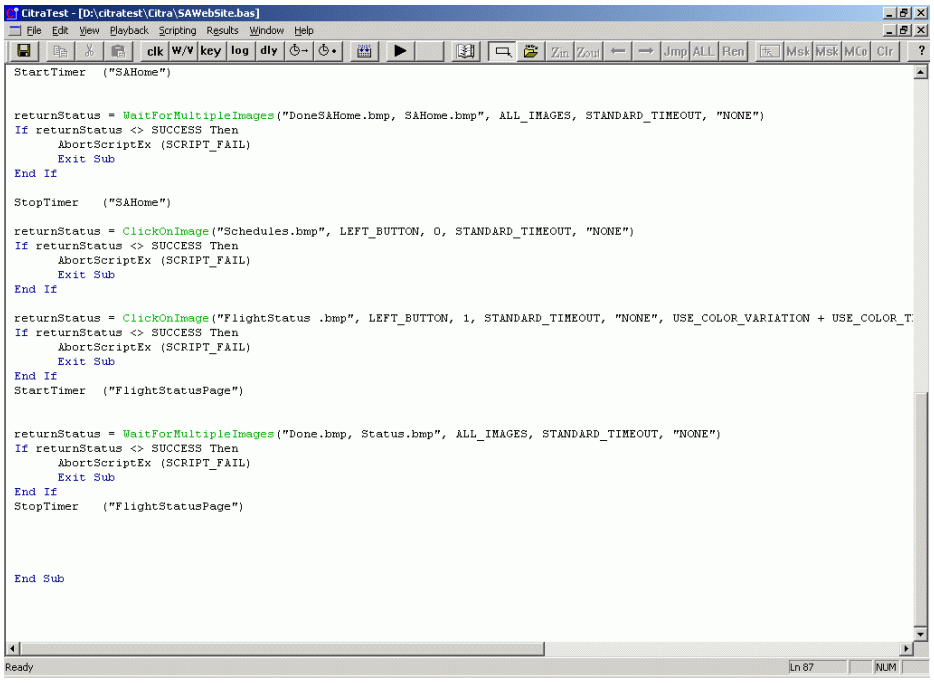

Figure 2.49: Setting a timer for the Flight Status page

62. The script should then attempt to query the availability information pertaining to a specific flight using the **Flight Status** page. For that, the script should first enter a specific

Flight Number in this page. To achieve this, first, capture an image of a small portion of the **Flight Number** field. Figure 2.50 reveals that the image has been captured and has been saved as **Flightnumber.bmp**.

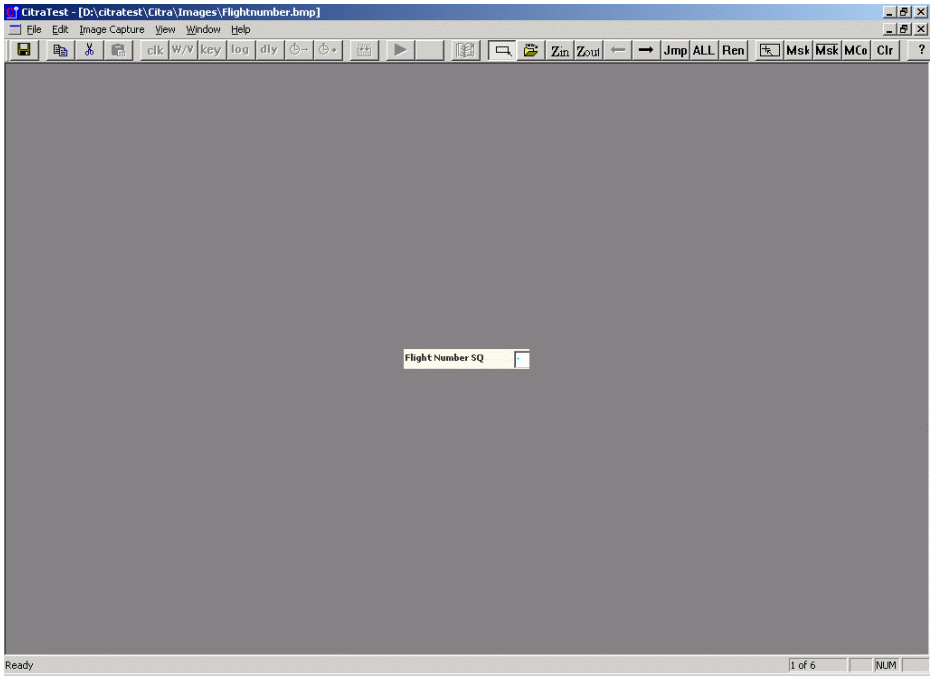

Figure 2.50: Saving the image of the Flight Number field

63. Prior to keying in the flight number, the cursor should be positioned in the **Flight Number** text box. To achieve this, associate a click event with the **Flightnumber.bmp** image (see Figure 2.51).

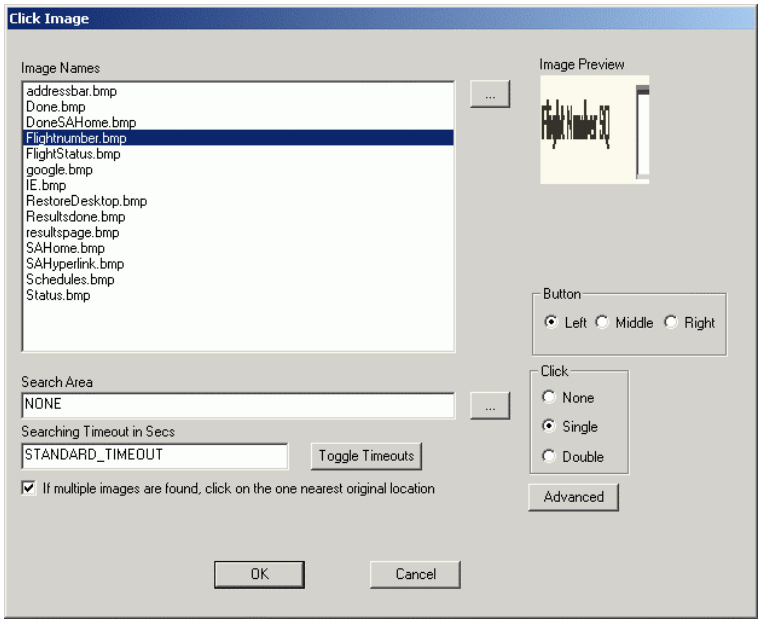

Figure 2.51: Clicking on the Flightnumber.bmp

64. To key in the flight number, click on the **key** button on the tool bar of the script window, and then specify a flight number in the **Text String** text box of Figure 2.52. In our example, *409* is the flight number. Once the text is specified, click on the **Add to Script Code >>** button corresponding to the text box, to generate the relevant script code for the keystrokes (see Figure 2.52). This script code will be added to the **Keystroke Script Code** list of Figure 2.52.

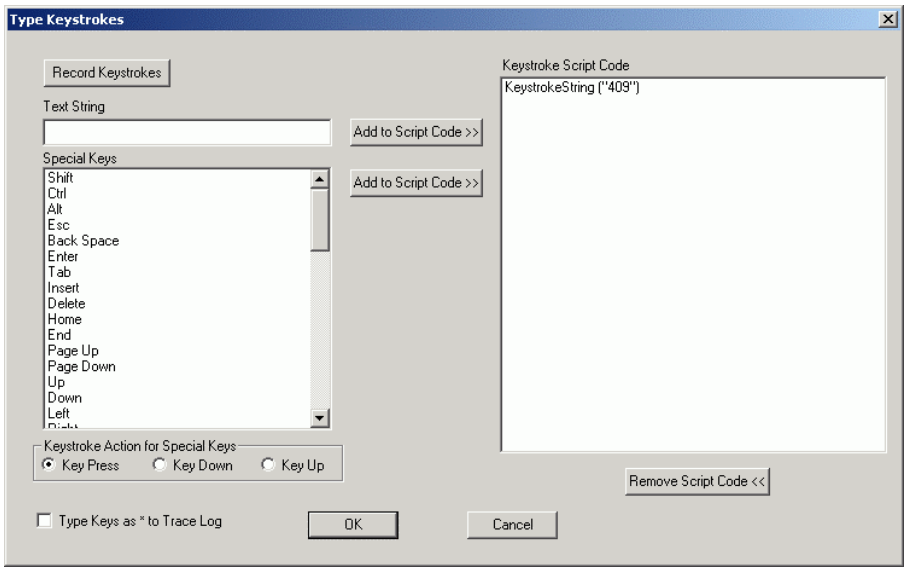

Figure 2.52: Typing the flight number to be queried

65. Then, click the **OK** button in Figure 2.52.

66. Our example seeks to figure out at what time the flight number *409* leaves *Chennai* on the current date. Therefore, the **Departing from** radio button has to be selected next. Since this option is selected by default, proceed to select the city from which the flight *409* departs. In other words, ensure that the script selects *Chennai* from the **City** list box. To click on the **City** list box, first, capture the whole/part of its image as depicted by Figure 2.53.

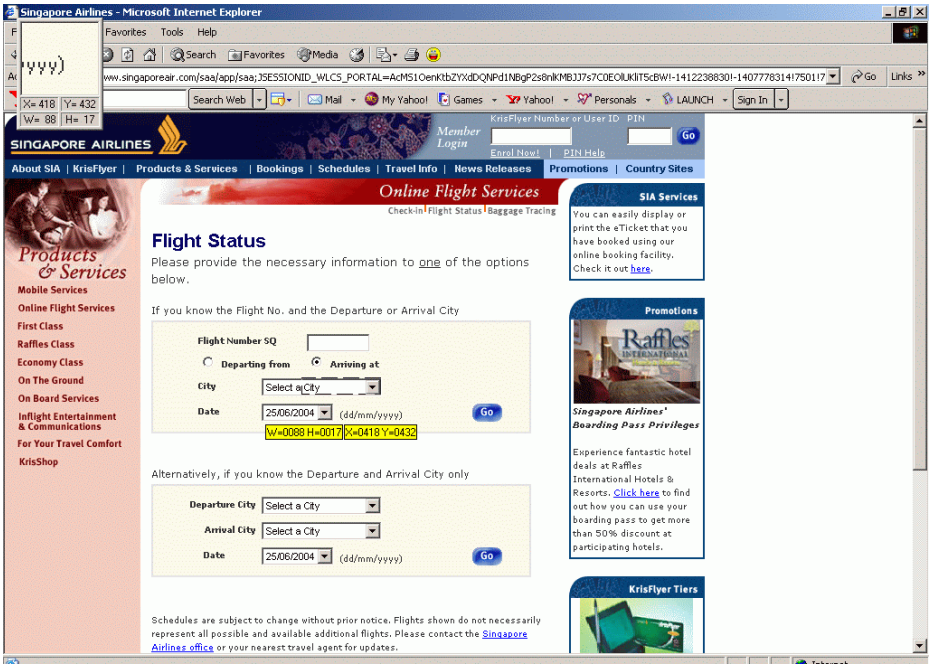

Figure 2.53: Capturing an image of the City list box

67. The captured image has been named as **CitySelect.bmp** (see Figure 2.54). Figure 2.54 also indicates the default click spot of the image.

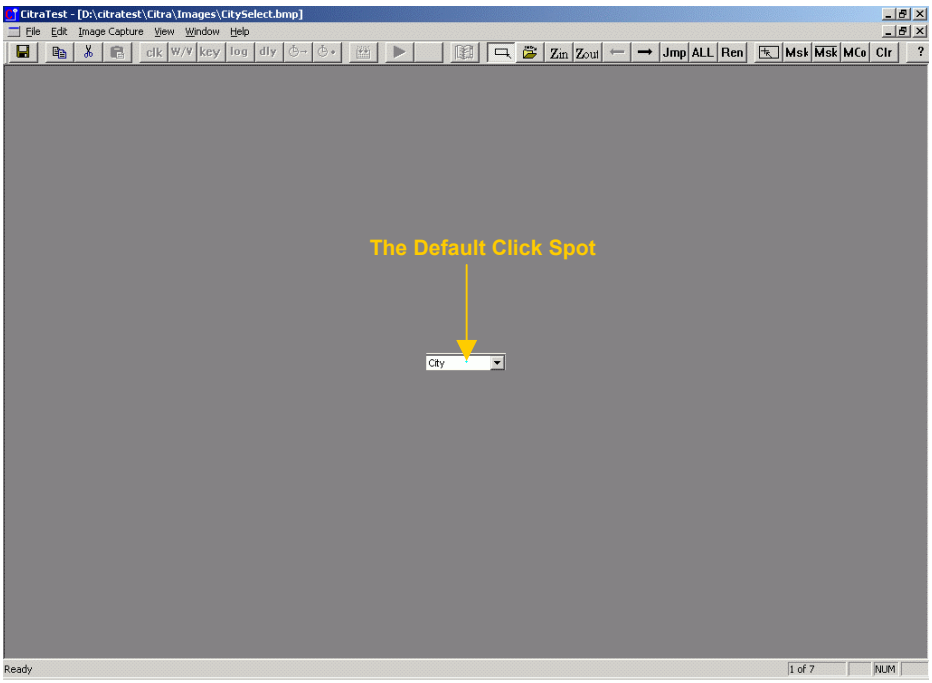

Figure 2.54: The default click spot of the CitySelect.bmp

68. To be on the safer side, let us position the click spot on the down arrow at the end of the **CitySelect.bmp** image. To achieve this, click on the button on the tool bar of Figure

2.55, and then, click on the down arrow at the left corner of the **CitySelect** image. Figure 2.55 indicates the new click spot.

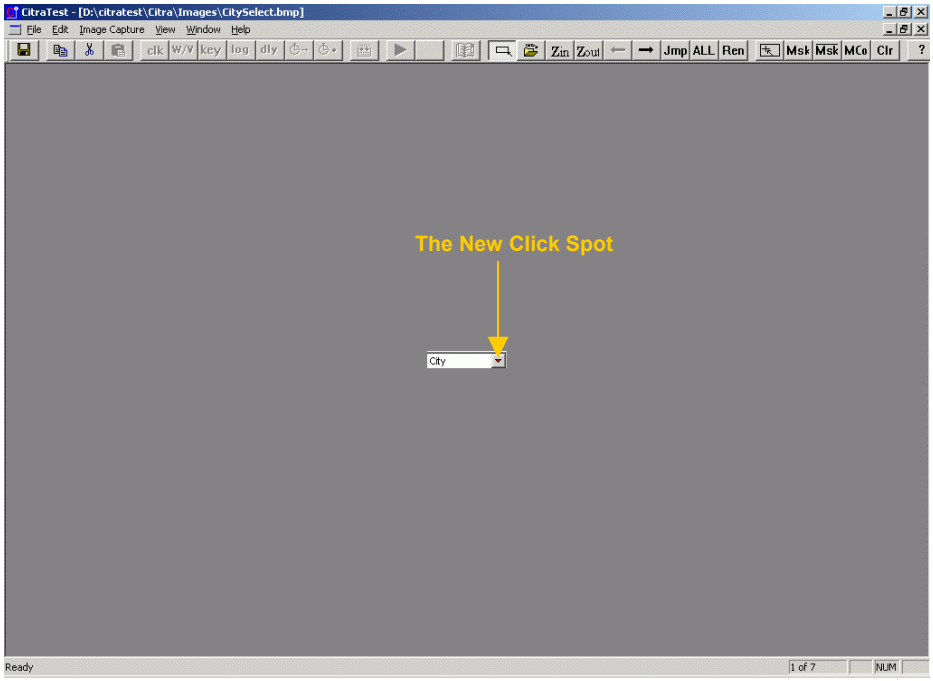

Figure 2.55: The new click spot for the CitySelect.bmp

69. Now, instruct the script to click on the **CitySelect.bmp**. To do this, click on the **clk** button on the tool bar of the script window, select **CitySelect.bmp** from the **Image Names** list of Figure 2.56, and then, click the **OK** button therein.

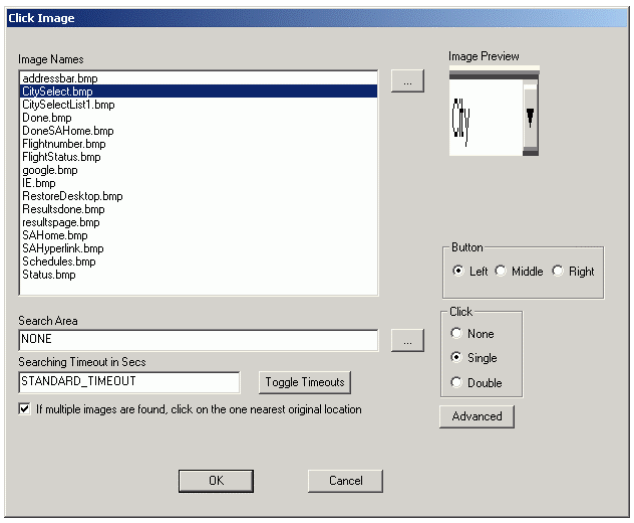

Figure 2.56: Clicking on CitySelect.bmp

70. Upon clicking the **CitySelect.bmp**, a drop-down list of cities will appear. The quickest way of getting to the *Chennai* option in this list would be to press **c** using your keyboard, and

then pressing the **Page Down** key. This will take you nearer to the *Chennai* option. You can then capture an image of the option and associate a click event with it.

Let us begin by recording the key strokes. For that, click on the **key** button on the script window's tool bar, and enter **c** in the **Text String** text box. Then, click on the **Add to Script Code >>** button next to the text box to generate the corresponding script code.

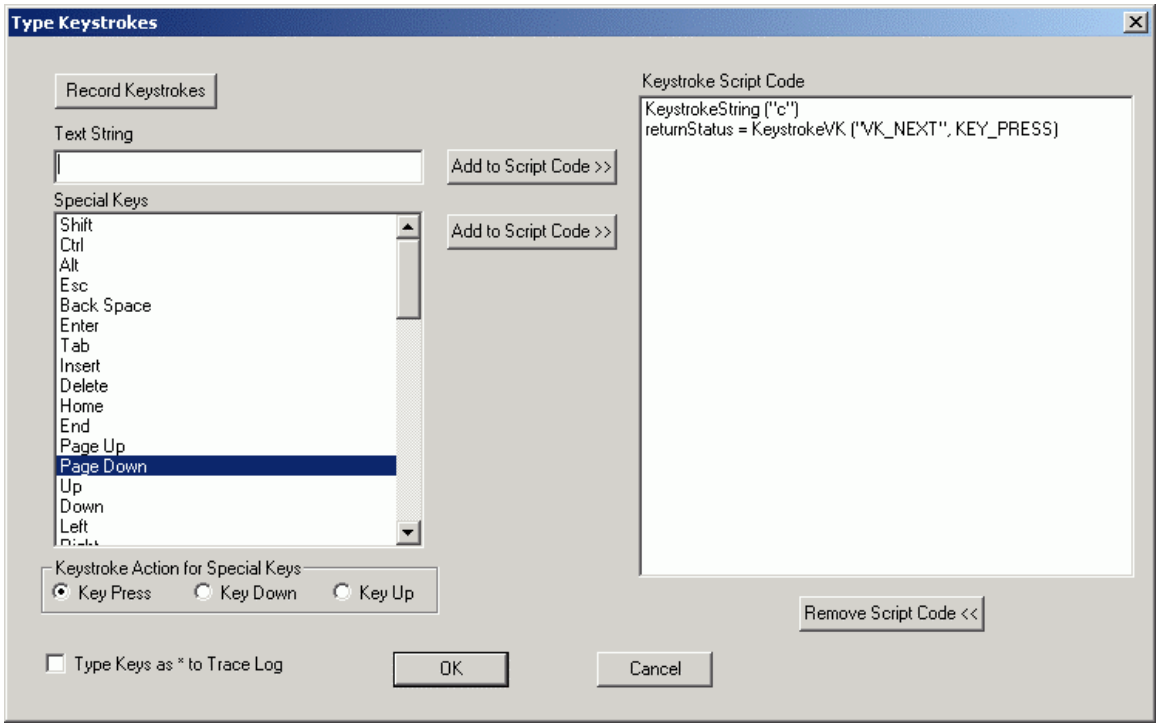

Figure 2.57: Entering 'n' to navigate to the cities beginning with the letter N

71. Next, select the **Page Down** key from the **Special Keys** list and add it to the **Keystroke Script Code** list by clicking on the **Add to Script Code >>** button (see Figure 2.57).

72. To click on the *Chennai* option, first, capture an image of the option as depicted by Figure 2.58.

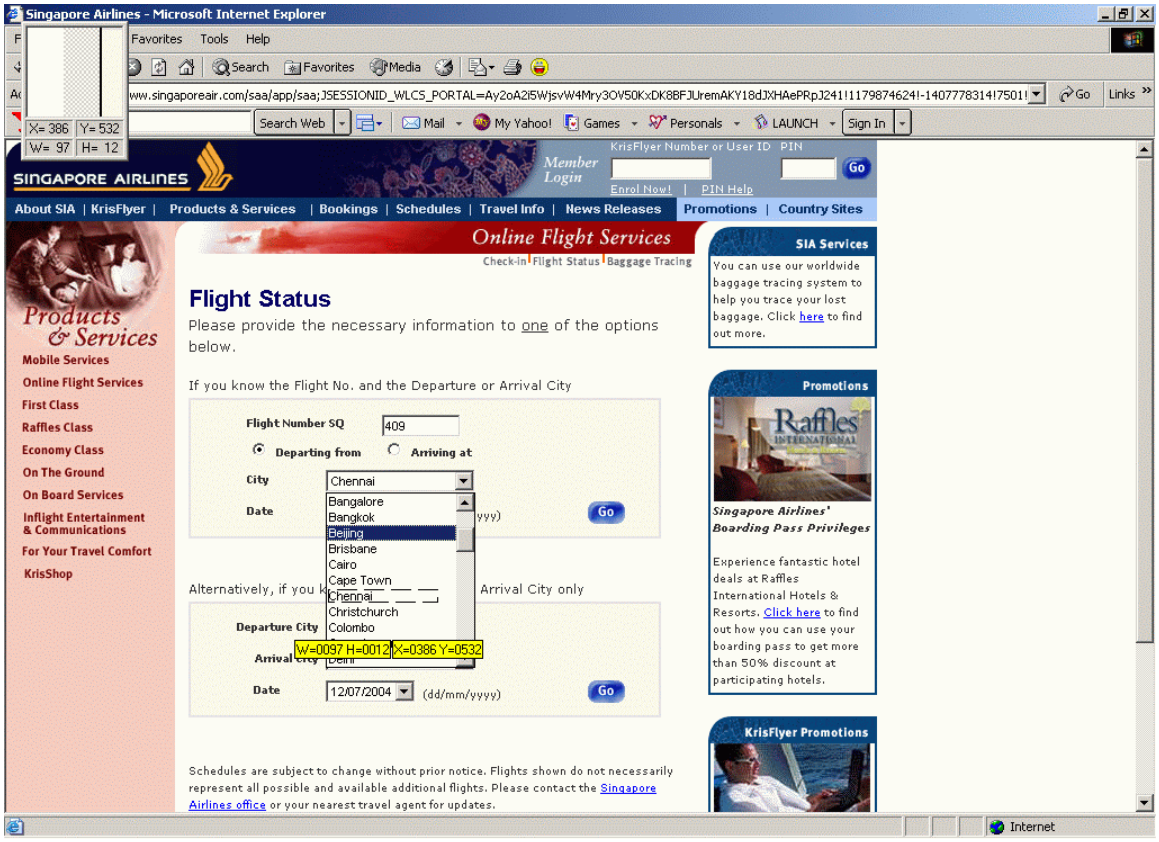

Figure 2.58: Capturing an image of the Chennai option

73. Save the image as **chennai.bmp**, and associate a click event with it. The VB script will get updated accordingly.

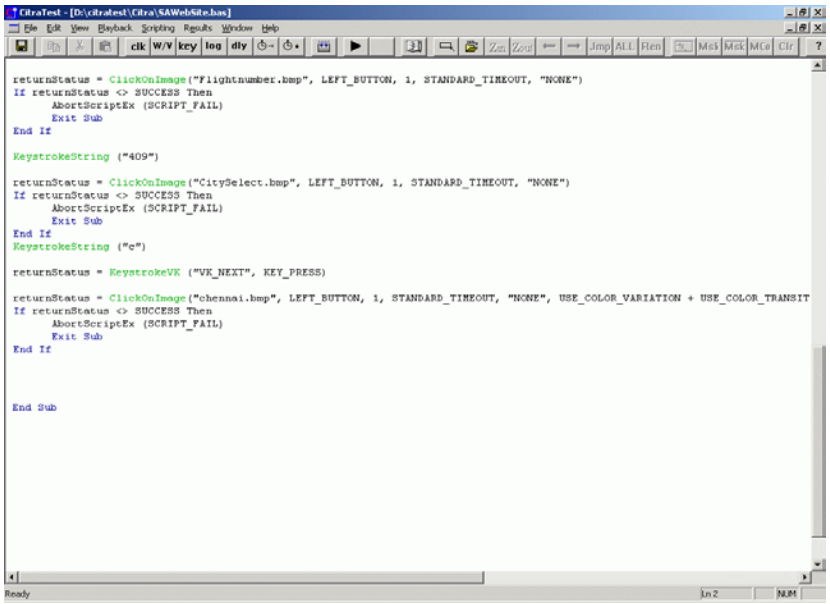

Figure 2.59: Script for clicking on the chennai.bmp

74. By default, the current date will be displayed against the **Date** field in the **Flight Status** page. This setting therefore, need not be disturbed. Hence, let us proceed to click on the **GO** button.

75. In situations where a page element to be clicked on is very similar (in both name and appearance) to another element in the same page, the probability of a script failure is very high, as the test script might fail to differentiate between the two elements. This is where the concept of a **Relative Search Area** comes into play. **Relative Search Areas** are used to limit the image and text recognition logic to a specified region of the screen. When a specific region in a page is set as a **Relative Search Area**, and this area is assigned to an image to be clicked on, then, during playback, the script will search only this region for the image. In our example, note that the **Flight Status** page consists of two **GO** buttons (see Figure 2.58). In order to enable the script to tell one from the other, let us define a **Relative Search Area**. The on-screen position of relative search areas, during script playback, is tied to a "relative reference" anchor image. Therefore, let us begin by capturing the anchor image.

76. **An anchor image should be from the same page as the relative search area.** Any of the existing (already captured) images from that page can be used for this purpose. For our example however, let us capture a new anchor image in the manner depicted by Figure 2.60.

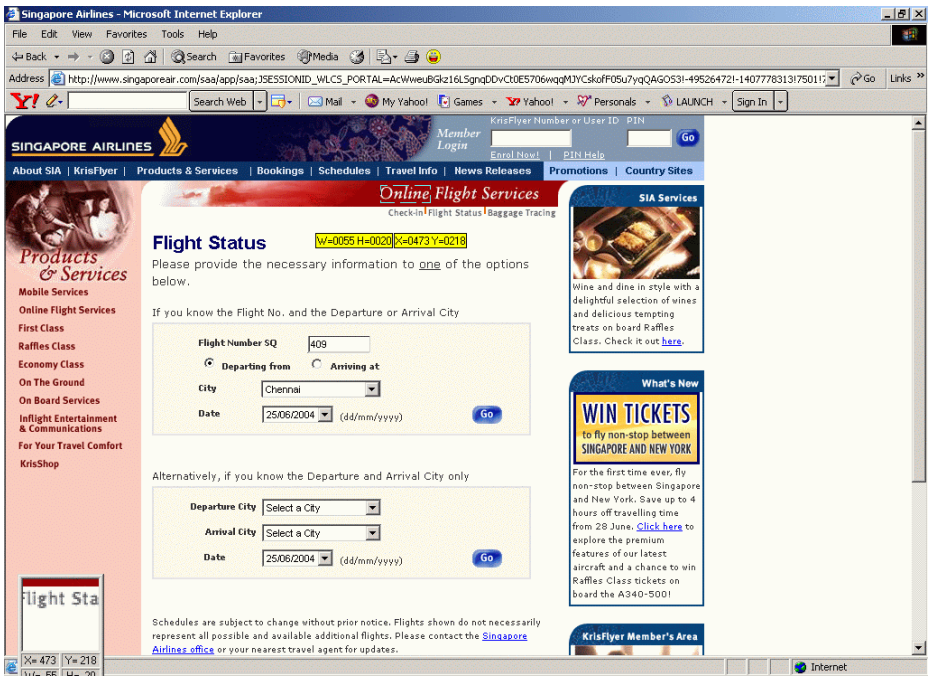

Figure 2.60: Capturing an anchor image

77. Next, save the image as **Anchor.bmp** (see Figure 2.61).

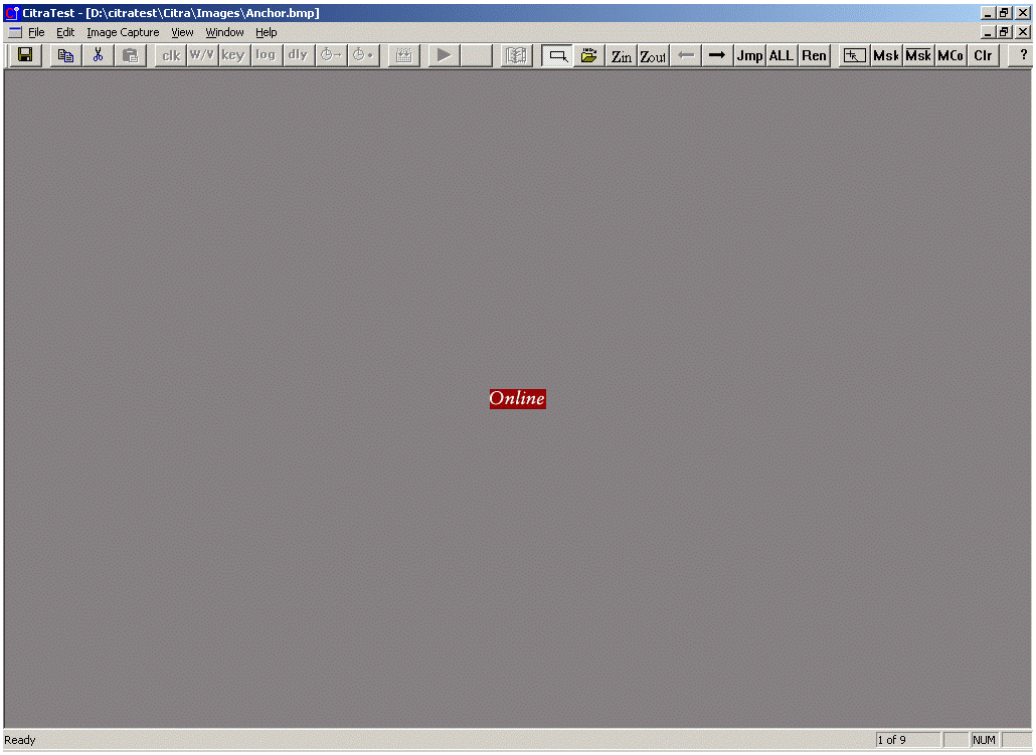

Figure 2.61: Saving the image as Anchor.bmp

78. Next, proceed to capture an image of the search area as depicted by Figure 2.62.

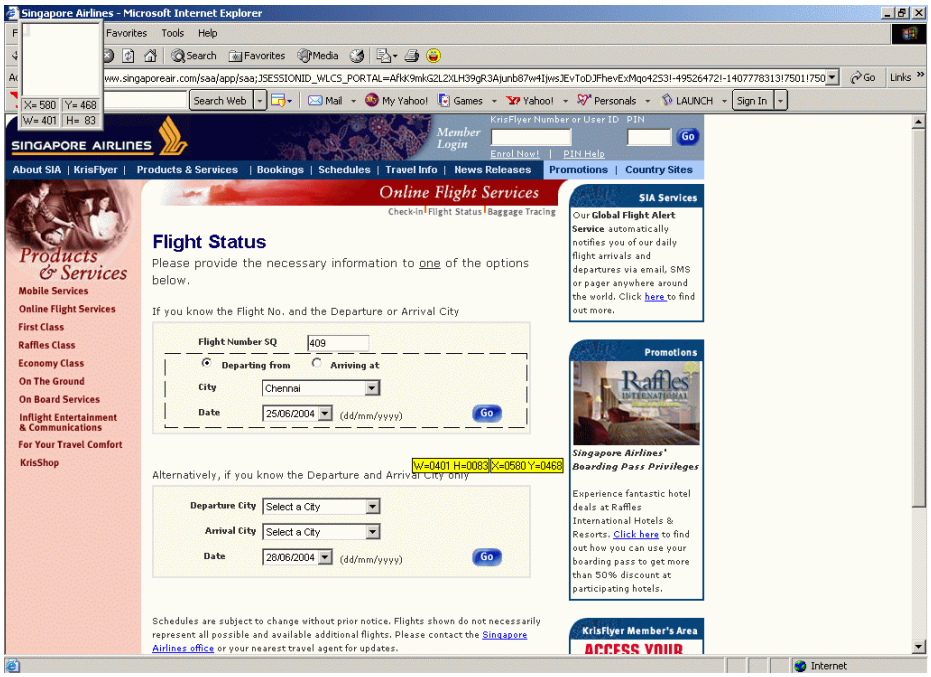

Figure 2.62: Capturing an image of the Relative Search Area

Note:

While capturing a relative search area, ensure that the image to be clicked on is also captured as part of the area (see Figure 2.62). In our example, this is the **GO** button.

79. Once the image is captured on to the image window, instead of saving it as a bitmap, select the File -> Save As -> Relative Area option to save it as a relative search area (see Figure 2.63

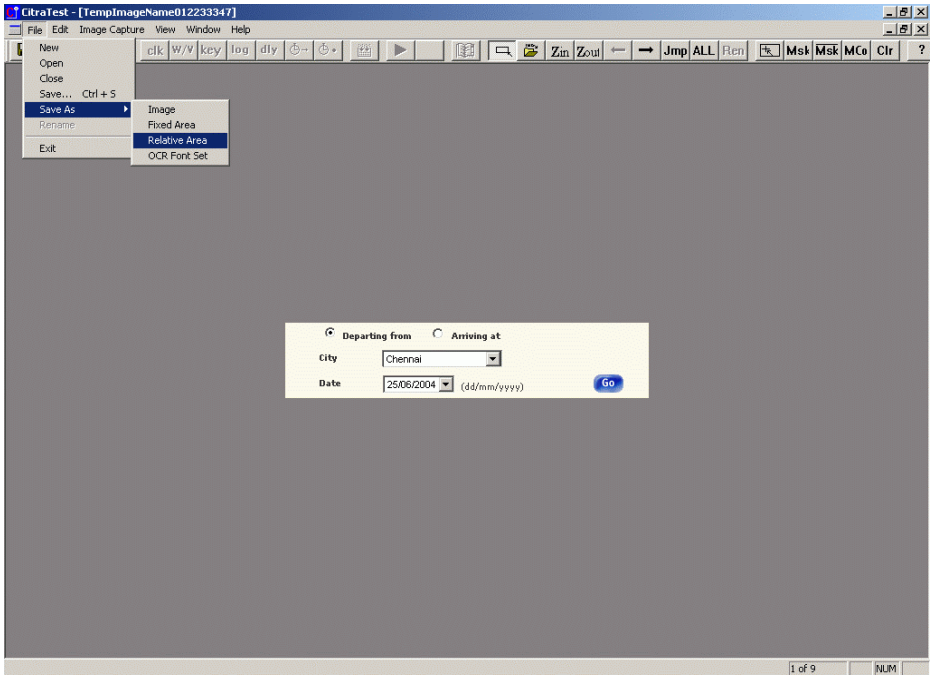

Figure 2.63: Saving the captured image as a Relative Search Area

80. Figure 2.64 will then appear wherein you need to specify a name for the relative search area, and bind it to the anchor image, **Anchor.bmp** (see Figure 2.64).

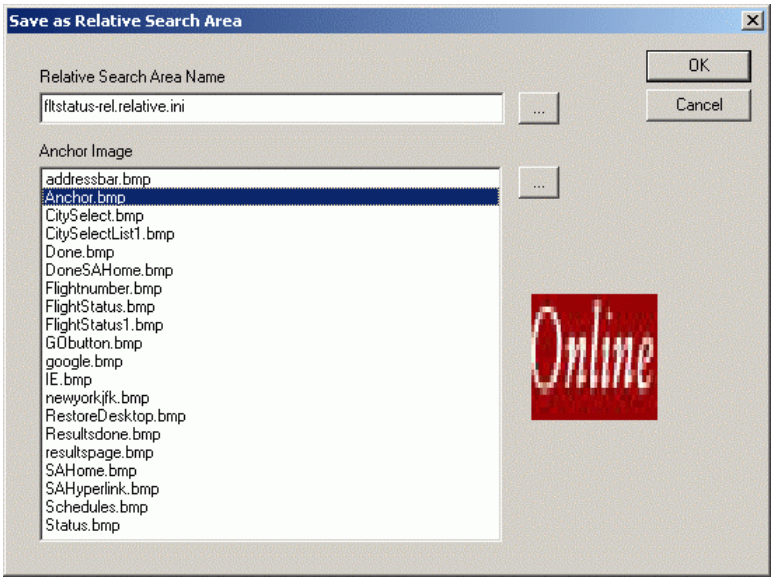

Figure 2.64: Binding a relative search area to the anchor image

81. Finally, click the **OK** button in Figure 2.64.

82. Then, proceed to click on the **GO** button, by first, capturing an image of the **GO** button (see Figure 2.65).

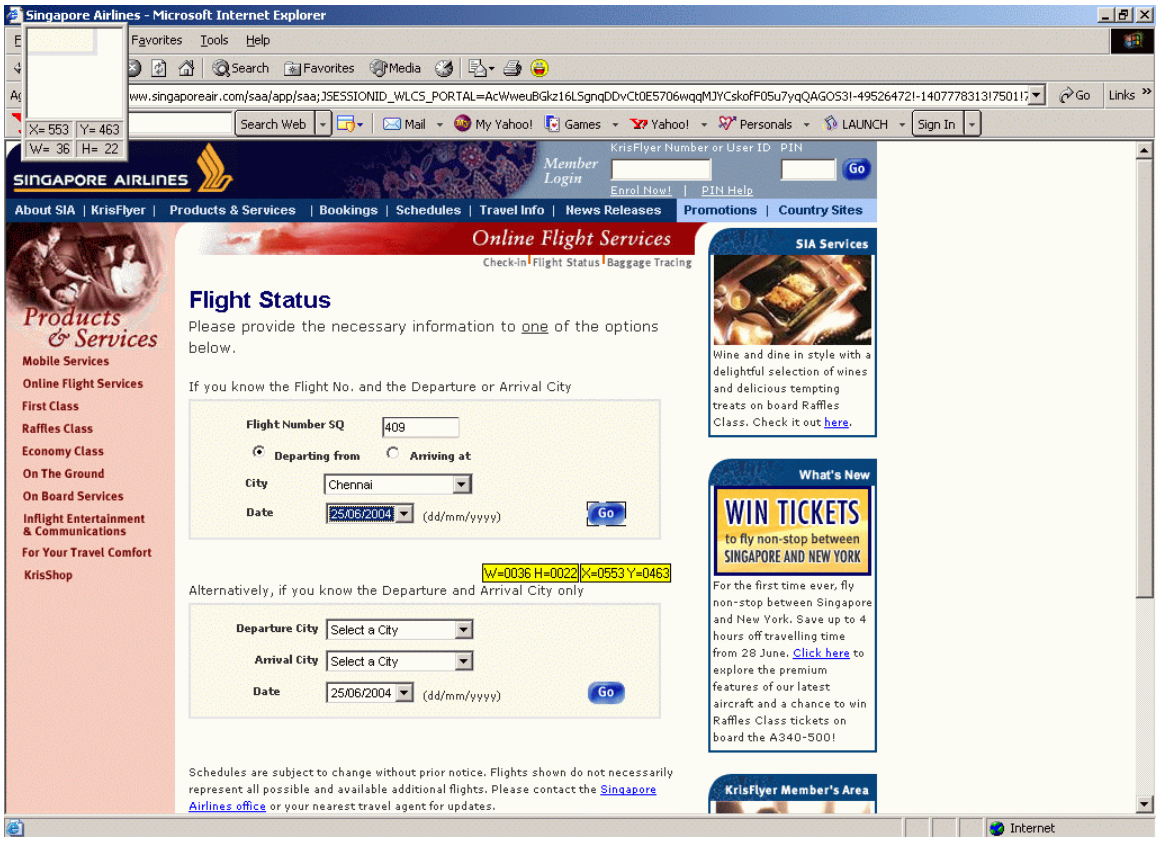

Figure 2.65: Capturing an image of the GO button

83. Save the image as **GObutton.bmp** (see Figure 2.66).

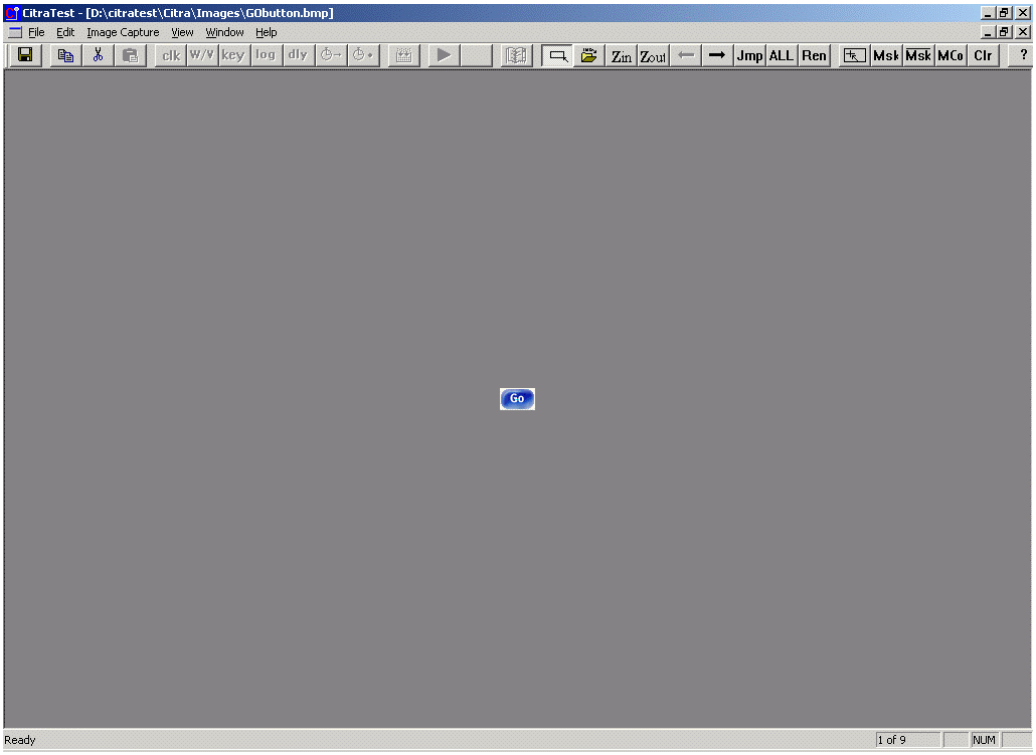

Figure 2.66: Saving the GO button image as GObutton.bmp

84. Then, return to the script window, and click on the **clk** button on its tool bar. From Figure 2.67 that appears, select the **GObutton.bmp** from the **Image Names** list, and specify the full path to its relative search area in the **Search Area** text box. By default, relative search areas will be saved to the **Images** directory only. Finally, click on the **OK** button in Figure 2.67.

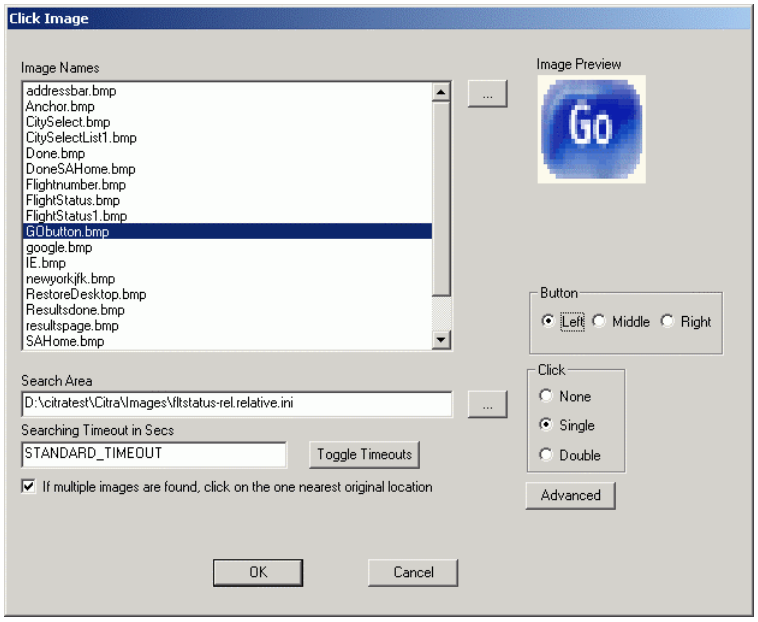

Figure 2.67: Clicking on the GO button

85. The script window will then display the corresponding script (see Figure 2.68).

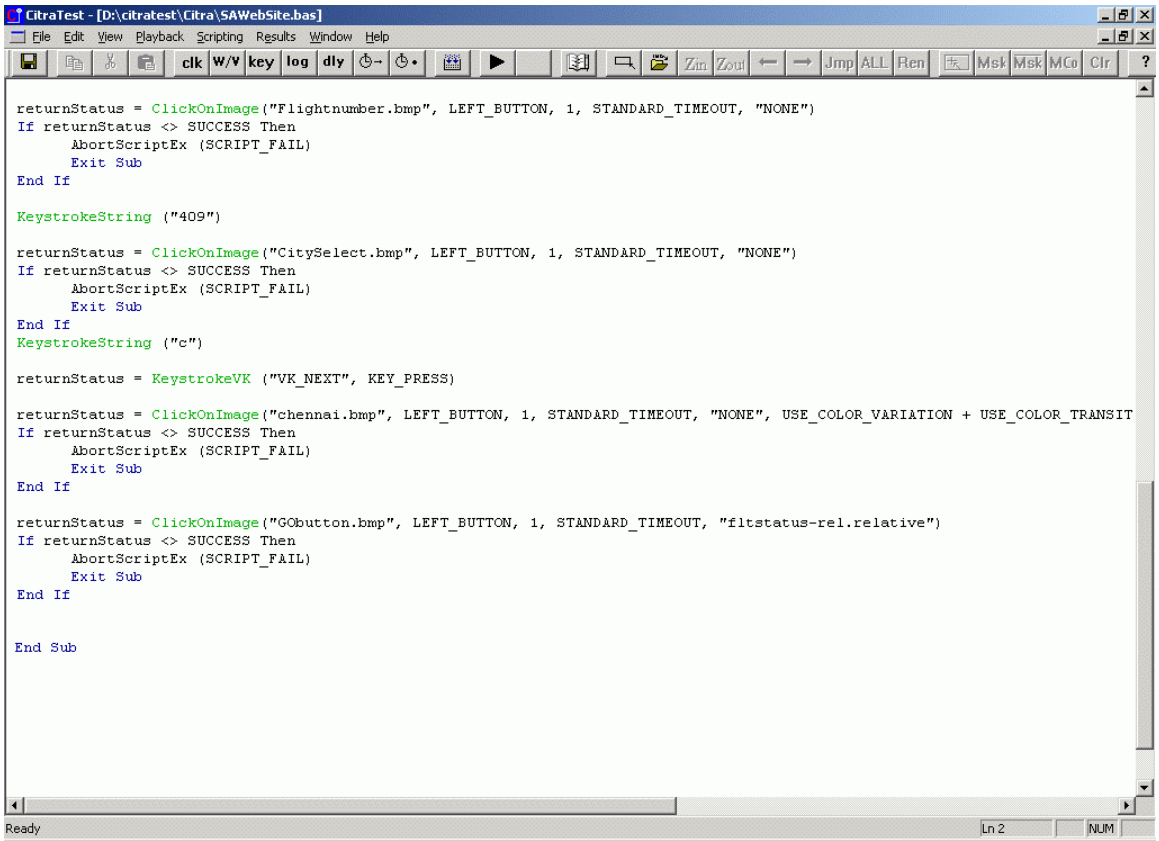

Figure 2.68: Script for clicking on the GO button

86. Once the **GO** button is clicked on, the status of the specified flight will be made available to you. To calculate the time taken for the query to execute, first, instruct the script to 'wait for' the title of the results page to appear. Therefore, proceed to capture the page title as depicted by Figure 2.69.

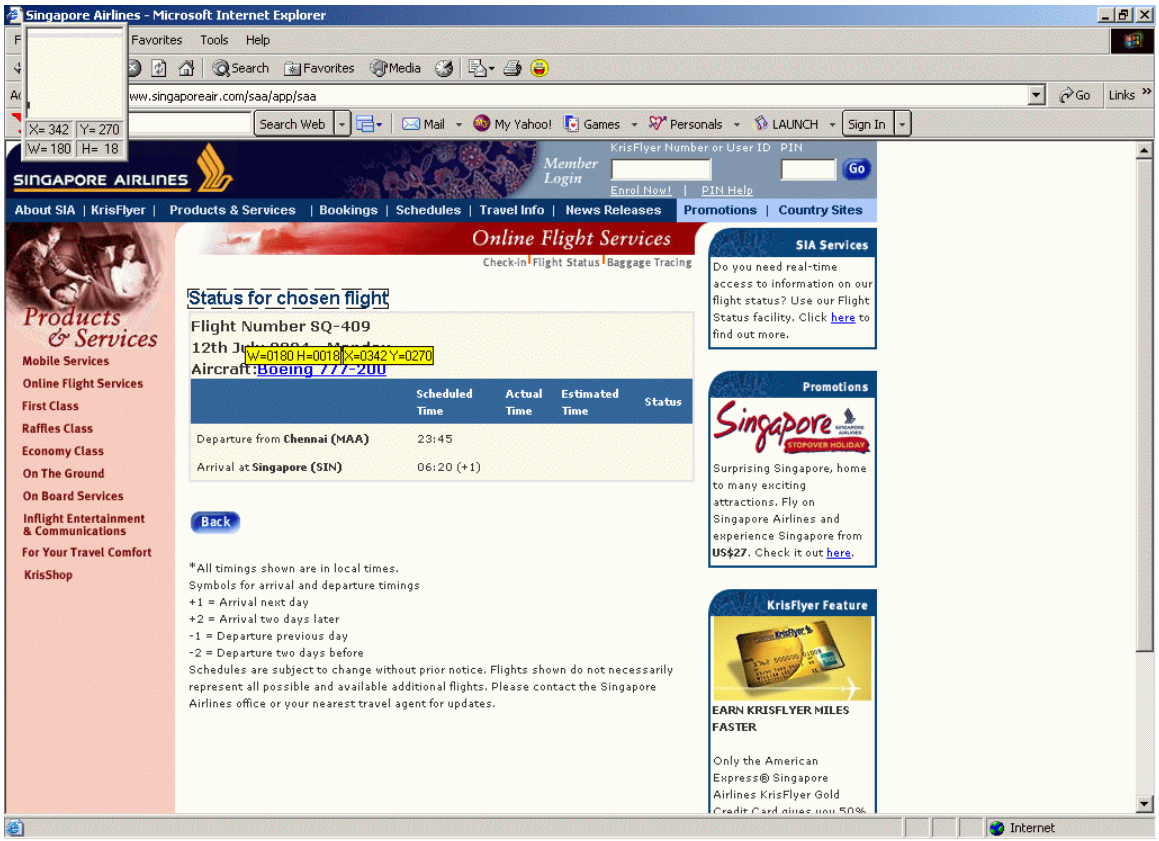

Figure 2.69: Capturing an image of the results page

87. Save the image as **StatusResult.bmp**. Next, navigate to the script window to build a script that will wait for the results page to appear. To do this, click on the **W / V** button on the tool bar, select the **StatusResult.bmp** from the **Image Names** list of Figure 2.70, select the **Wait For ANY Image** option, and finally, click the **OK** button.

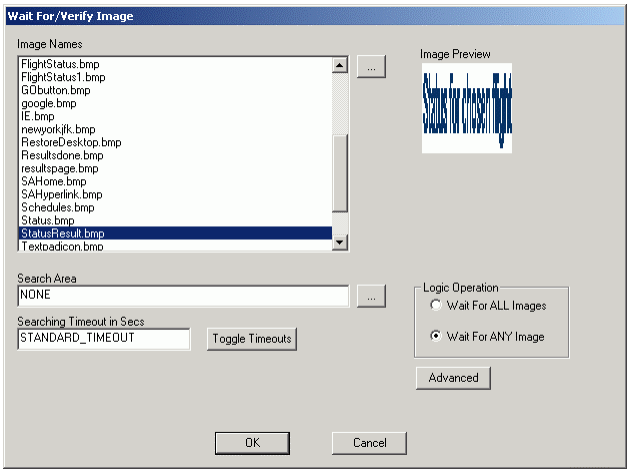

Figure 2.70: Waiting for the StatusResult.bmp to appear

88. The script to the above effect will then appear in the script window. Then, start and stop a timer named *FlightStatusCheck* to record the query execution time. The *StartTimer* code should be inserted before this "wait for" script, and the *StopTimer* code should be inserted after the "wait for" script (see Figure 2.71).

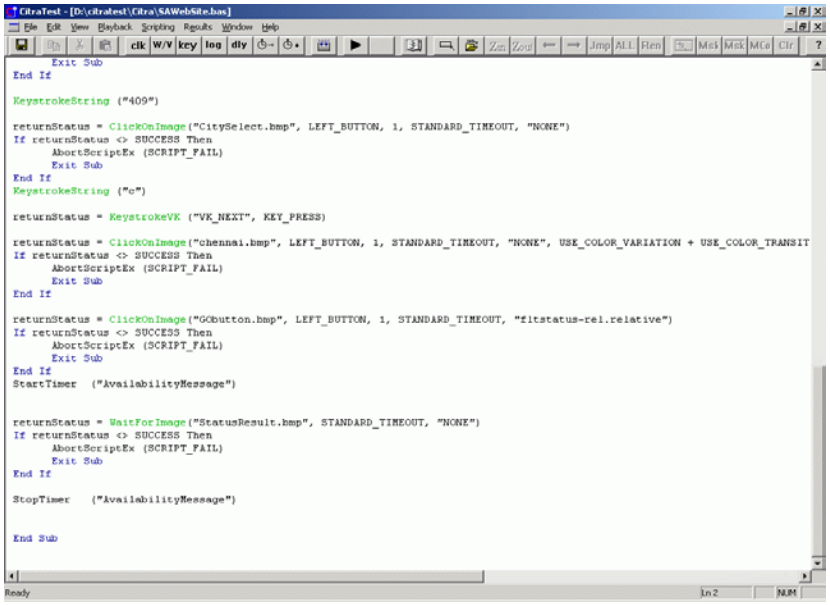

Figure 2.71: Script for recording the query execution time

89. With that, request emulation is complete. Now, compile the script by clicking the button on the tool bar of Figure 2.71. Once compilation becomes successful, run the script by clicking on the button.

Note:

You are advised to compile and execute the test script at various stages of development, so that issues can be detected early and attacked easily.

Note:

Upon successful execution of the script, a message box informing you of the success will appear. You are advised to disable this message box and restore the desktop instead. To change the playback settings to reflect this, select **Playback Options** from the **Playback** menu on the script window.

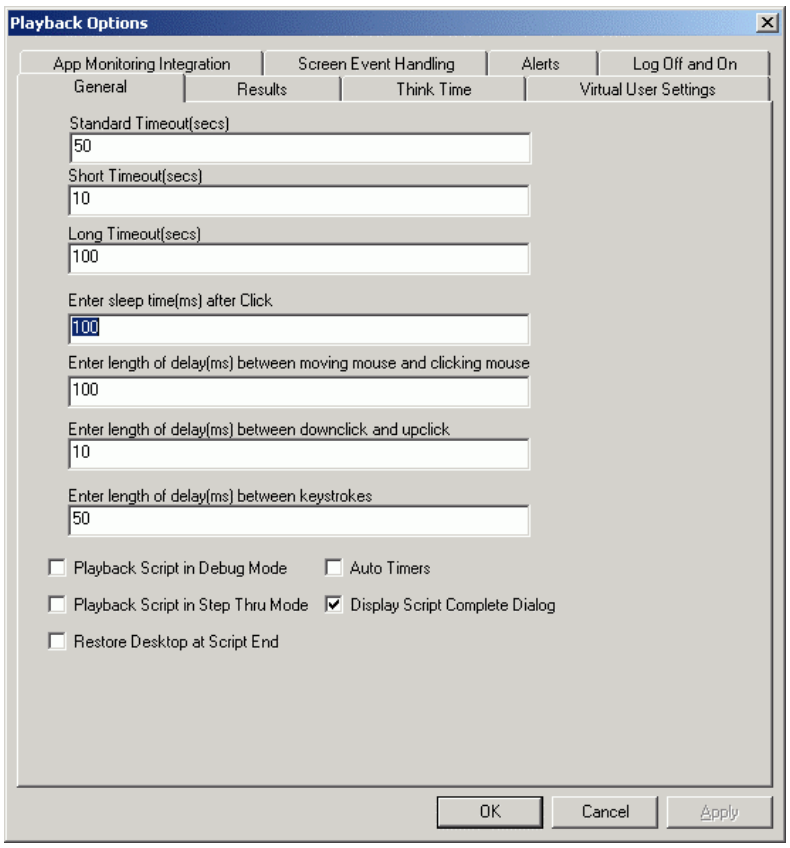

Figure 2.72: Playback options

In the **General** tab that opens by default (see Figure 2.72), deselect the **Display Script Complete Dialog** check box, and enable the **Restore Desktop at Script End** check box. Finally, click the **OK** button.

Similarly, by default, the **STANDARD_TIMEOUT** period is set to 50 seconds. This default setting can be changed using the **General** tab of Figure 2.72.

2.2.2 Configuring the eG manager to Work with CitraTest

The next step is to configure the eG manager to work with CitraTest. The eG manager supports an *Emulated Client* component type that has been specifically designed to extract performance metrics from request emulators such as CitraTest.

To configure the eG manager to work with CitraTest, do the following:

1. Login to the eG manager as *admin* with password *admin*.
2. Next, add the component to be monitored as an *Emulated Client*. Therefore, first, select the **Add/Modify Servers** option from the **Infrastructure** menu, and select the *Emulated Client* option from the **Component type** list box (see Figure 2.73).

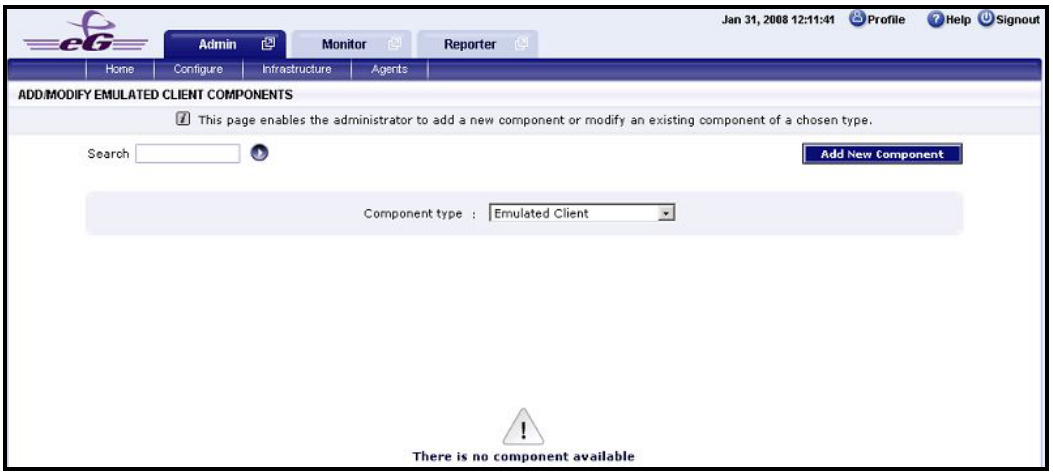

Figure 2.73: Selecting the Emulated Client component type

3. Then, click on the **Add New Component** button in Figure 2.73 to add a new component of type *Emulated Client*.
4. Provide the IP address and host name of the component in Figure 2.74 that appears. In our example, the component to be monitored is the web server, 192.168.10.32:80, which hosts the **Singapore Airlines** web site. Therefore, provide 192.168.10.32 against the **Host IP** and the **Host/Nick name** text boxes, and 80 against **Port number**. Then, select the external agent that will execute the **SAWebsite** script, and click the **Add** button therein to register the changes. In our example, the external agent monitoring the web server (192.168.10.32:80) is, 192.168.10.41.

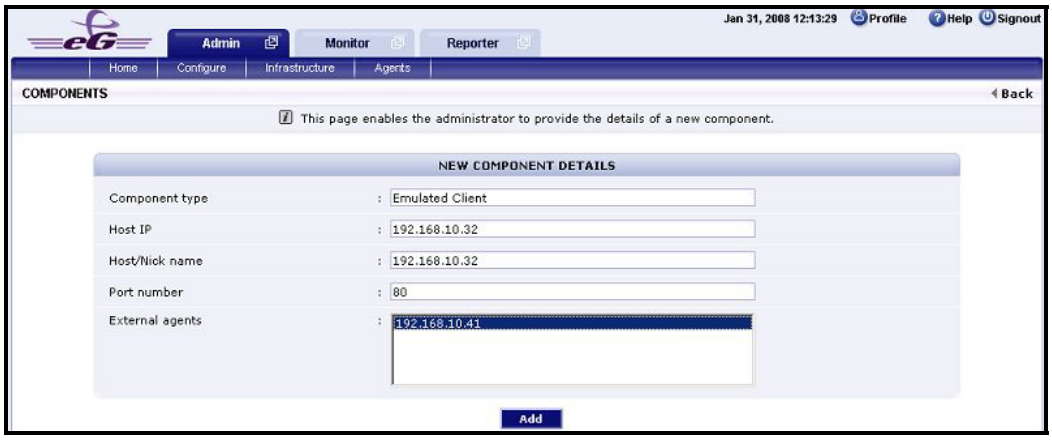

Figure 2.74: Adding a component of type *Emulated Client*

Note:

Only those external agents for which the **CLIENT EMULATION** capability has been enabled will be displayed in the **EXTERNAL AGENTS** list of Figure 2.74. Such agents cannot monitor any other component type.

5. Now, try to sign out of the eG administrative interface by clicking on the **SIGNOUT** button at the top left corner of the screen. Upon attempting to sign out, a **LIST OF UNCONFIGURED TESTS** listing the tests that are still to be configured will appear (see Figure 2.75).

Figure 2.75: List of tests to be configured

6. Figure 2.75 reveals that the **CitraClientEmulation** test associated with the *Emulated Client* component type is yet to be configured. This test reports the availability and response time of the application being monitored by the CitraTest tool. To configure this test, click on the test name in Figure 2.75. This will open Figure 2.76.

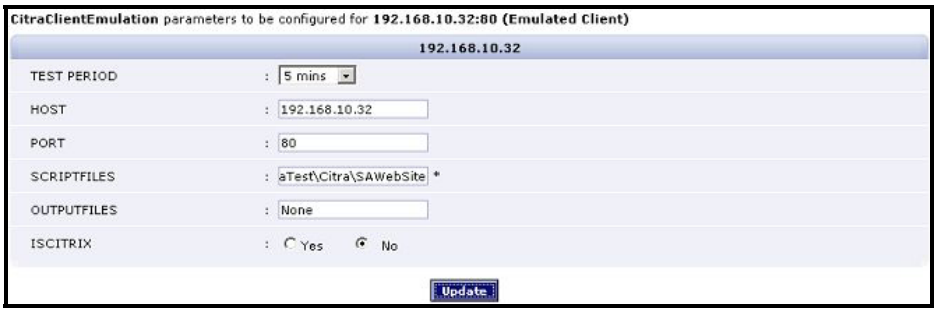

Figure 2.76: Configuring the CitraClientEmulation test

7. Specify the following in Figure 2.76:

   a. **TEST PERIOD** – How often should the test be executed

   b. **HOST** – The host on which the test will run. In our example, the test will attempt to extract measures from the host, 192.168.10.32.

   c. **PORT** – The port at which the specified **HOST** listens. In our example, this is port 80.

   d. **SCRIPTFILES** – The full path to the script file that is to be played back for emulating a request to, and extracting metrics from the monitored application. Multiple script files can be provided as a comma-separated list, but all script files should monitor the same application only. In our example, the path to the **SAWebSite** script has to be specified here.

   > **Note:**
   >
   > If the script file resides on another host, then ensure that the location of the script file is mapped to any drive on the measurement host.

   e. **OUTPUTFILES** – Enter the full path to the output file that contains the metrics extracted by the specified script file. Here again, multiple output files can be provided as a comma-separated list, but only if multiple script files are also provided.

Note:

- If **None** is specified here, then the eG system will collect statistics from the default output files associated with each of the specified script files. The default output files will be present in the same location as the respective script files, and will have the same name as the script files. In our example, the value of the **OUTPUTFILES** parameter can remain as **None**.
- While specifying multiple output files, ensure that they are provided in the same order as their corresponding script files in the **SCRIPTFILES** text box.
- If the **SCRIPTFILES** parameter consists of multiple entries and the **OUTPUTFILES** parameter consists of only one, then eG will automatically associate the first script file entry in the **SCRIPTFILES** box with the **OUTPUTFILES** entry. Measures pertaining to the other script files will therefore not be displayed in the eG monitor interface.

f. **ISCITRIX** – If the specified script emulates a request to a Citrix client then specify **Yes** here. If not, specify **No**. Our example does not attempt to extract measures from a Citrix client. Therefore, provide **No** here.

8. Next, click on the **Update** button in Figure 2.76 to complete the configuration.
9. Now that CitraClientEmulation test has been configured, eG Enterprise will playback the specified **SCRIPTFILES** according to the chosen **TEST PERIOD**. Whenever the **SAWebSite** script is played back, the CitraTest tool will collect the availability and response time metrics for every *timer* configured in the script, and will store these details in the corresponding **OUTPUTFILES**. eG Enterprise will then extract the measures from the **OUTPUTFILES** and display them in the monitor interface.
10. To view the measures in the eG monitor interface, first, **SIGNOUT** of the administrative interface.

2.2.3 Starting the External Agent

Refer to the *eG Installation Guide* for an elaborate procedure on starting the eG agent on Windows environments. In our example, the external agent, 192.168.10.41, which is associated with the *EmulatedClient* component (192.168.10.32:80), will have to be started.

Before starting the external agent assigned to an *Emulated_client*, ensure that the *eGurkhaAgent* service is allowed to interact with the desktop. To do this, do the following:

1. Open the **Services** window by following the menu sequence: Start -> programs -> Administrative Tools -> Services (in a Windows NT environment), or Start -> Settings -> Control Panel -> Component Services (in a Windows 2000 Professional environment).
2. Right-click on the *eGurkhaAgent* service therein, and select the **Properties** option.

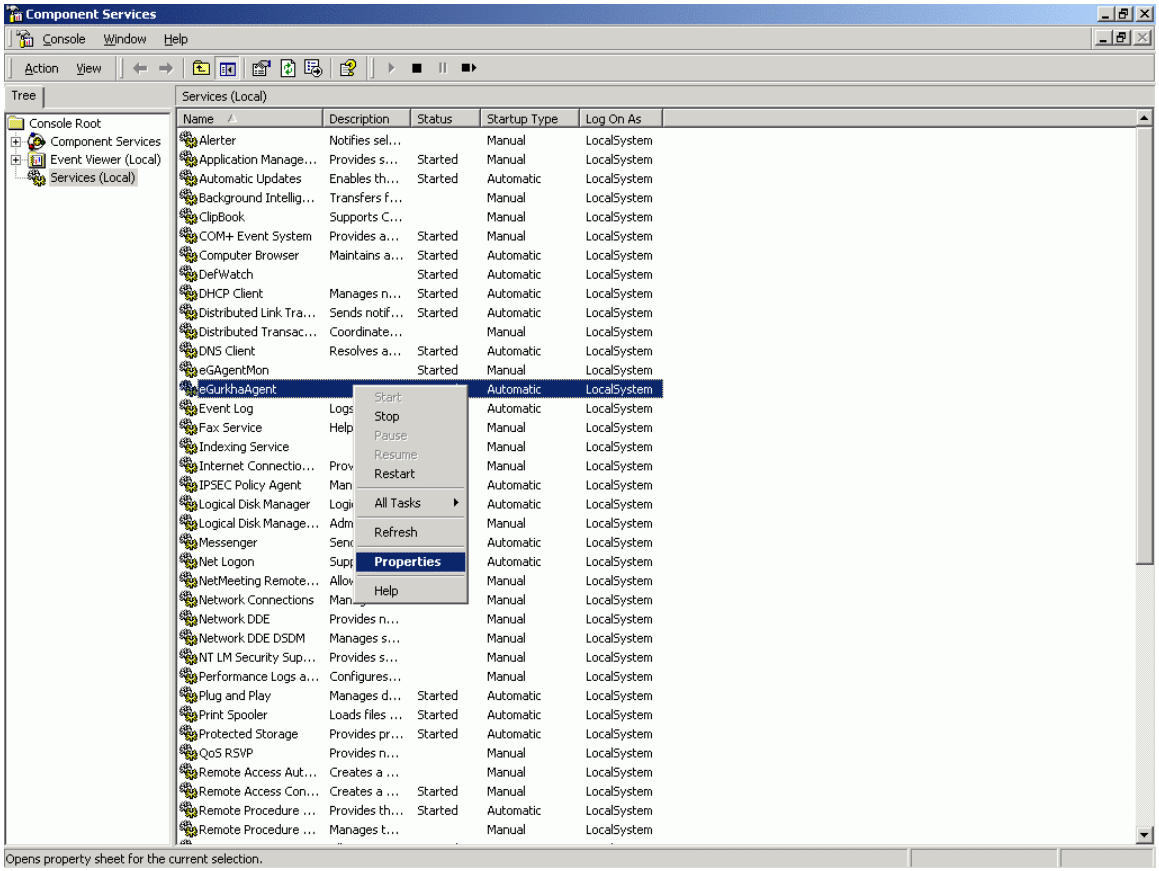

Figure 2.77: Selecting the Properties option

3. Click on the **Log On** tab in the **Properties** dialog box (see Figure 2.78), and select the **Allow service to interact with desktop** check box.

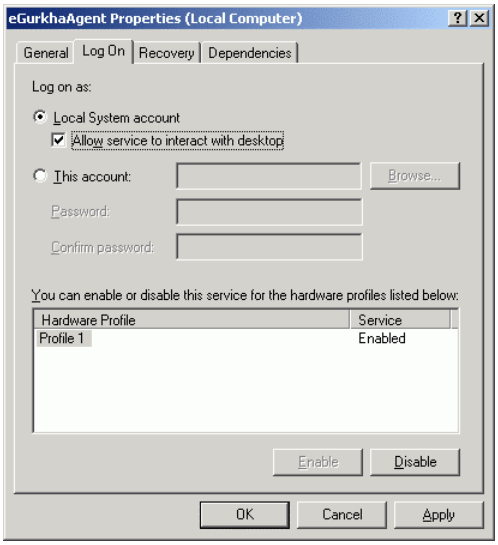

Figure 2.78: Allowing the service to interact with the desktop

4. Finally, click on the **Apply** button, and then the **OK** button.

2.2.4 Viewing the Measures

To view the measures reported by the **CitraClientEmulation** test, do the following:

1. Login to the eG monitor interface as *supermonitor* with password *supermonitor*.
2. From the **Components** menu, select the **Servers** option. Once the **COMPONENT LIST** page appears, select the *Emulated Client* option from the **Type** list therein, so as to view the current state of all components of type *Emulated Client*. Then, click the **Submit** button (see Figure 2.79).

Figure 2.79: The Independent Components page

3. The *Emulated Client* component that we had configured previously will then be listed (see Figure 2.79). Click on the component to view its layer model, tests, and measurements (see Figure 2.80).

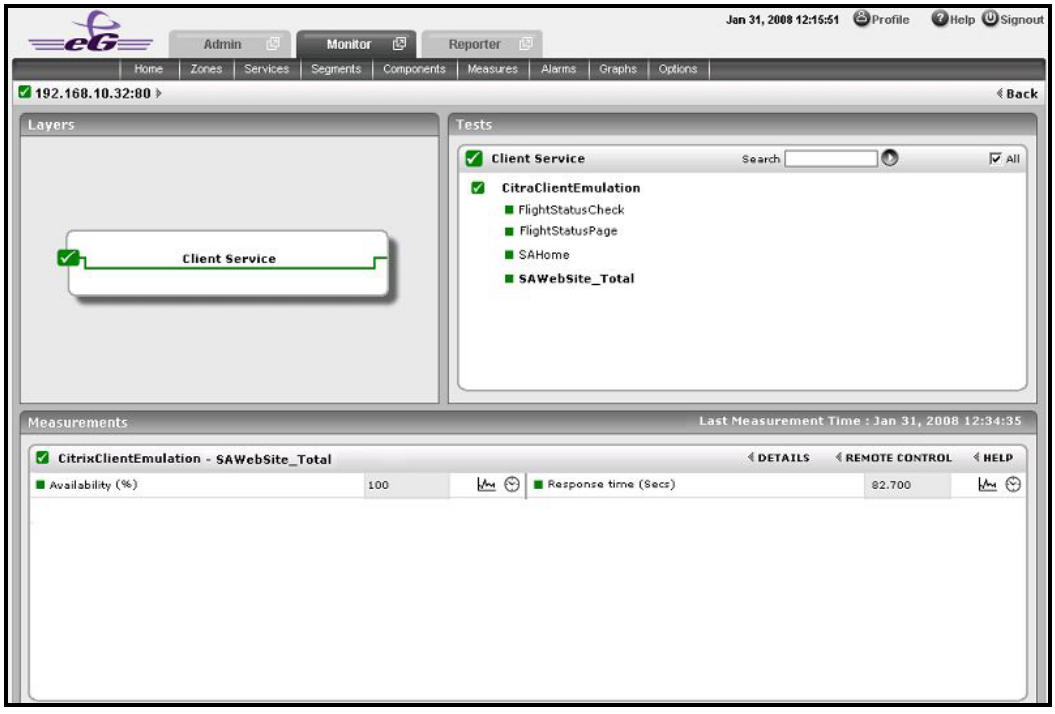

Figure 2.80: Viewing the layer model, tests, and measurements of the Emulated Client

4. A timer maps to a step in a multi-step service interaction. eG monitors steps using timers. From Figure 2.80, it is evident that all the steps that were monitored using the **SAWebSite** script appear as descriptors of the CitraClientEmulation test.

5. Besides the steps that we had explicitly added to the script, Figure 2.80 also displays an additional step, namely, *SAWebSite_Total*. This step tracks the time taken by every activity performed by the script, calculates the sum of the duration of the activities, and provides this sum as the response time of the monitored application. Figure 2.80 reveals the response time value returned by the *SAWebSite_Total* step. This is the total time taken to access the **Flight Status** page, request for **Flight Status** information, and receive a response from the web server.

6. To verify the correctness of the reported value, switch to the system hosting the CitraTest Development environment, and open the **SAWebSite** script in the script window. Then, open the Timer log by following the menu sequence View -> View Timer Log.

7. The **SAWebSite.timer.log** file of Figure 2.81 reveals the start time, end time, and the duration of every activity performed by the **SAWebSite** script. According to Figure 2.81, the duration of the timer *SAWebSite_Total* is *82.7 seconds*. Note that the eG monitor interface of Figure 2.80 reveals the same value.

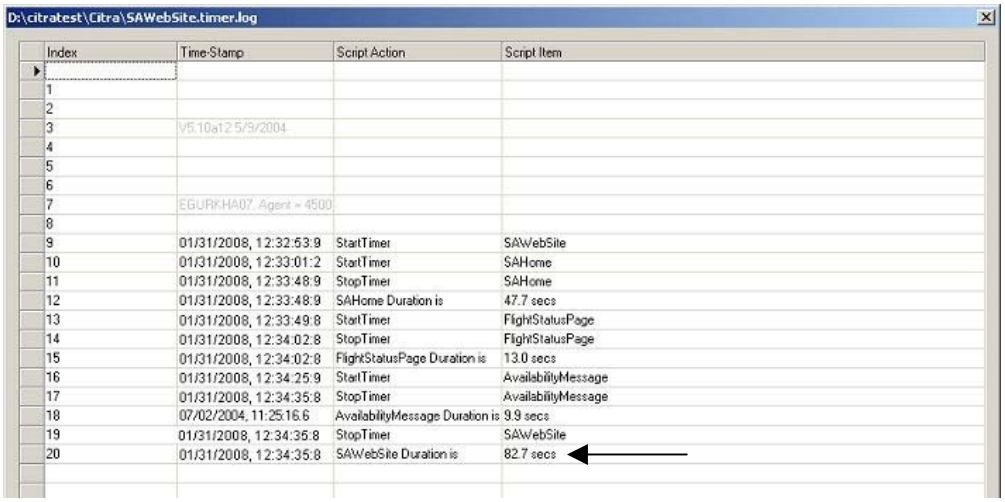

Figure 2.81: The timer log file of the SAWebSite script

The table below briefly explains the **Availability** and **Response time** measures returned by the CitraClientEmulation test.

Measure Name	Description
Availability (%)	This relates to the availability of the page, i.e whether a page has downloaded successfully or not. In Windows Applications, this relates to whether the step has executed properly or not. It is expressed in terms of percentage (%) – 100% if the download is a success and 0% if not. A variation of this is for the availability of all the steps in total - if even 1 step has failed, the total availability is 0 %.
Response time (Secs)	This relates to the time taken for the page to download or the step to execute. If the total availability is 0 % then the response time

	is set as 'unknown', else it is expressed in 'secs'.

2.3 Client Emulation for a Citrix Application

The second example consists of a Citrix farm, comprising of a Citrix server, 192.168.10.28:1494. Assume that a Textpad application has been published on this Citrix server. The script to be built should access the Citrix server through the web, and should try to open the Textpad application on it. In addition, the script should key in a few words in the Textpad and close the application without saving the changes.

2.3.1 Building a Script File

Let us now begin script building.

1. Create a new VB project named **CitrixEx** using steps 1-6 of Example 1.
2. Since the Citrix server has to be accessed through the web, once again, the Internet Explorer needs to be opened. Since these procedures have been dealt with elaborately in Example 1, let us look at what needs to be done once the IE window opens.
3. Now that the IE window is open, key in the URL of the Citrix client to connect to (192.168.10.28) in the **Address** box. Prior to that, the cursor must be positioned inside the **Address** box. To instruct the script to do the same, capture an image of the **Address** box, and associate a click event with it. This again, has been explained in great detail in Example 1. The next step therefore, is to type the URL. For that, open the script window, click on the **key** button on its tool bar, and enter http://192.168.10.28/ in the **Text String** text box (see Figure 2.82). Then, click on the **Add to Script Code >>** button adjacent to the text box to add the equivalent script code to the **Keystroke Script Code** box. Then, click the **OK** button in Figure 2.82.

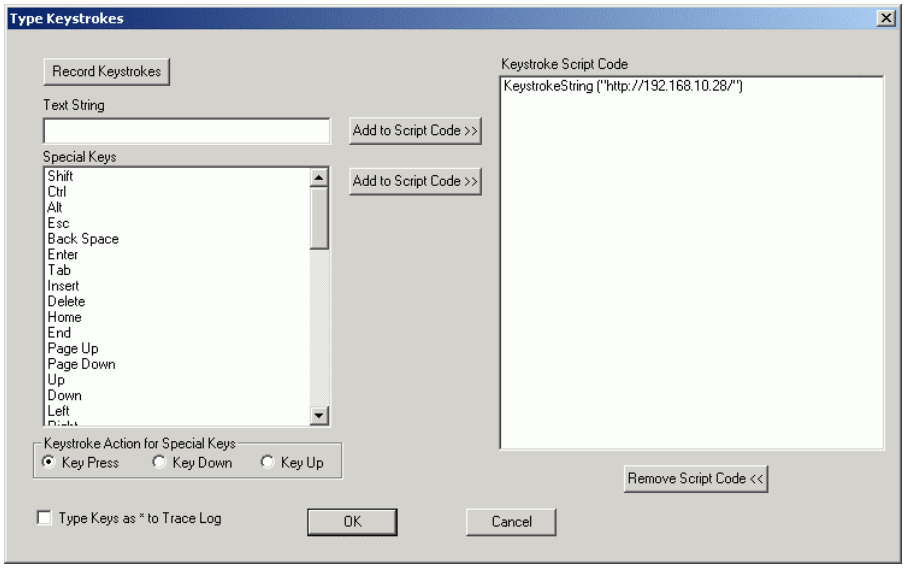

Figure 2.82: Keying in the URL of the Citrix client

4. Next, ensure that the script presses the **Enter** key on the keyboard (see Figure 2.83).

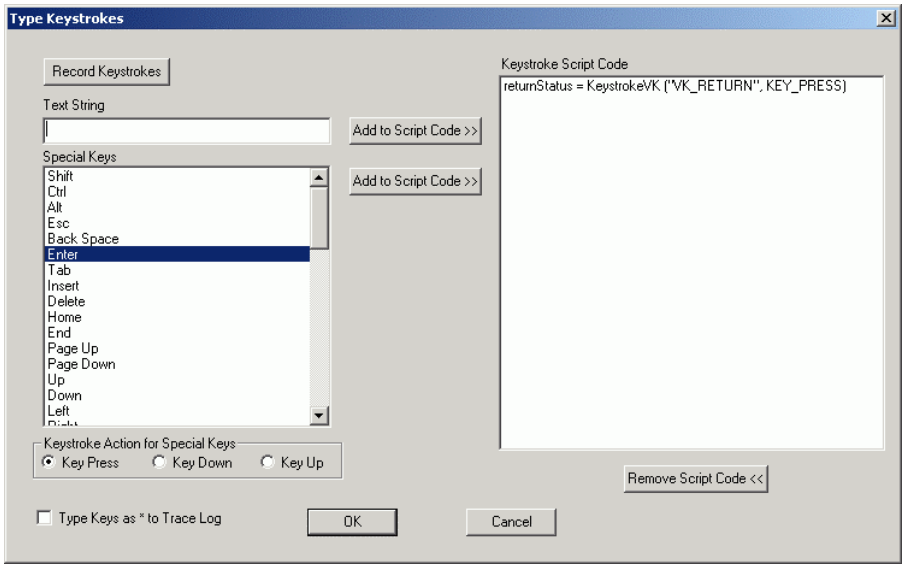

Figure 2.83: Pressing the Enter key after typing the URL

5. The script will get updated accordingly (see Figure 2.84).

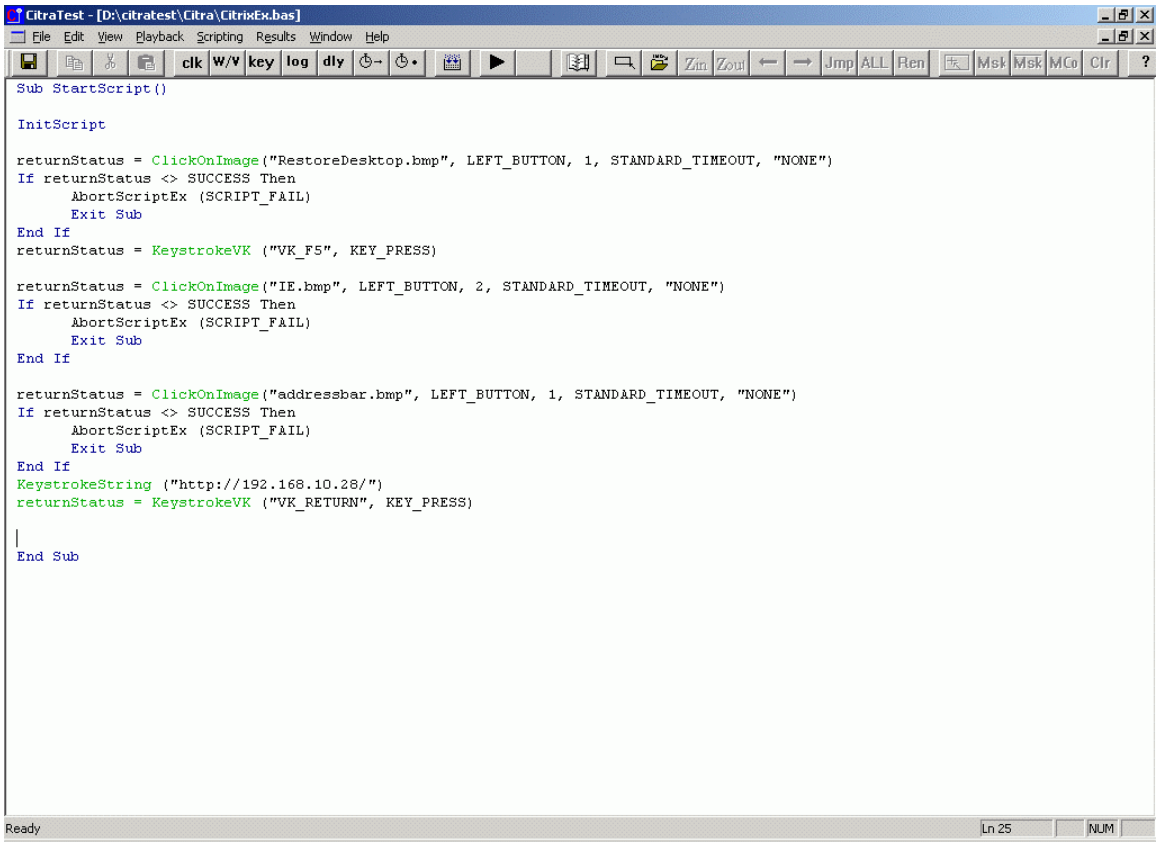

Figure 2.84: Script for typing the URL and pressing the Enter key

6. The Citrix client login page will then appear. To verify whether the page has downloaded completely, instruct the script to wait for the appearance of the Citrix logo on top of the

page, and the **Done** message on its status bar. Since the **Done** image has already been captured (in Example 1), proceed to capture the **Citrix logo** alone as depicted by Figure 2.85.

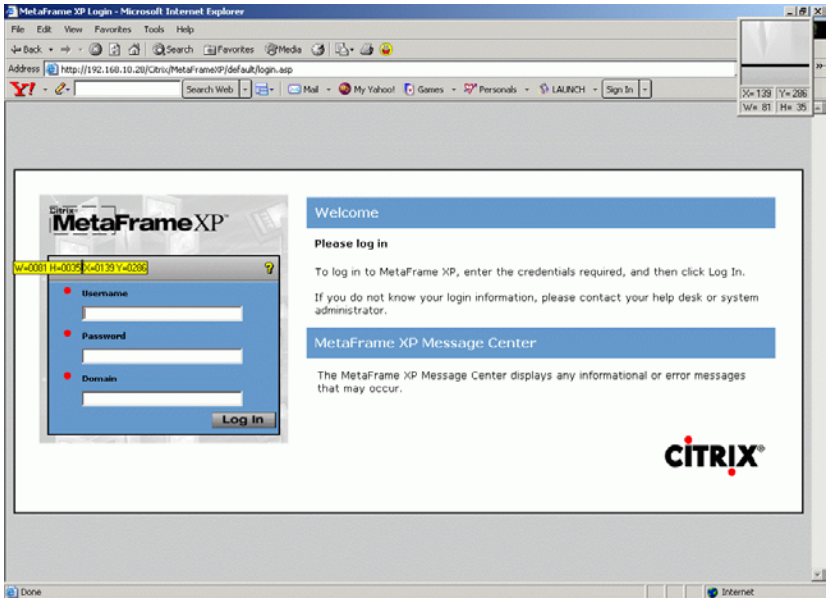

Figure 2.85: Capturing the Citrix log image

7. Save the image as **Citrixlogo.bmp** (see Figure 2.86).

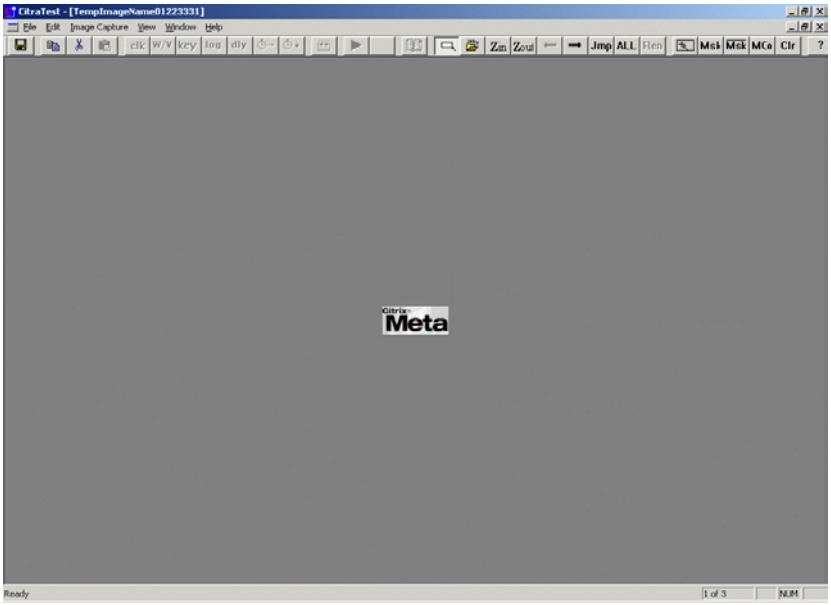

Figure 2.86: Saving the Citrix logo

8. Associate a 'wait for' event with both the **Done** and **Citrixlogo** bitmaps using the procedure depicted by Figure 2.87.

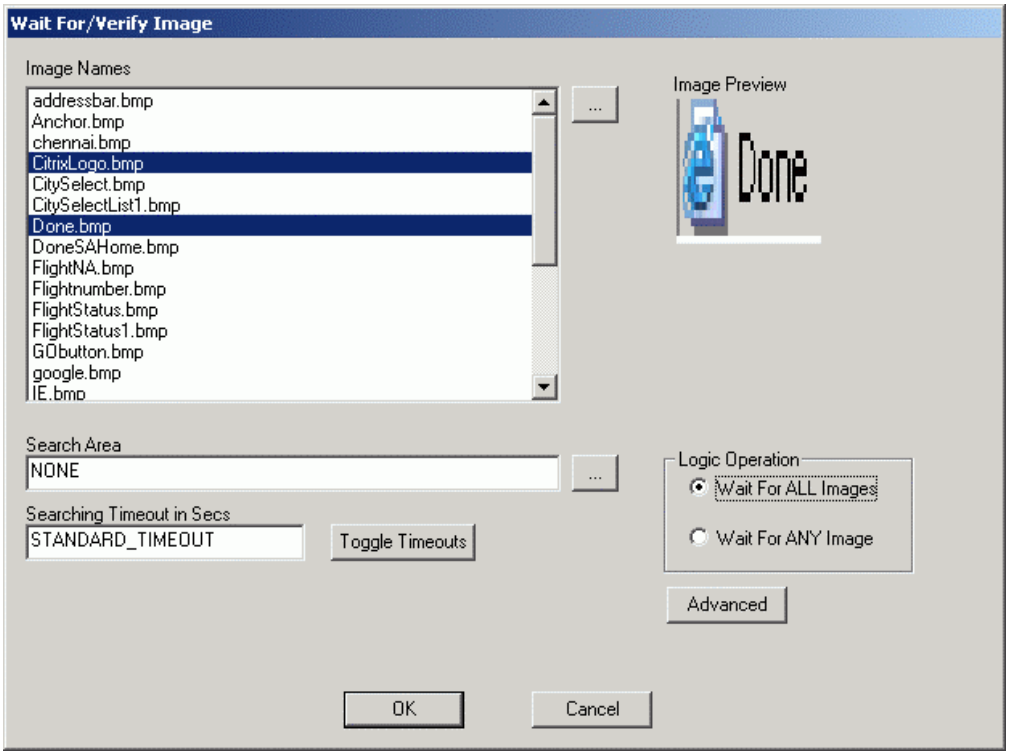

Figure 2.87: Waiting for the Done and Citrixlogo images

9. The script will reflect the recent changes (see Figure 2.88).

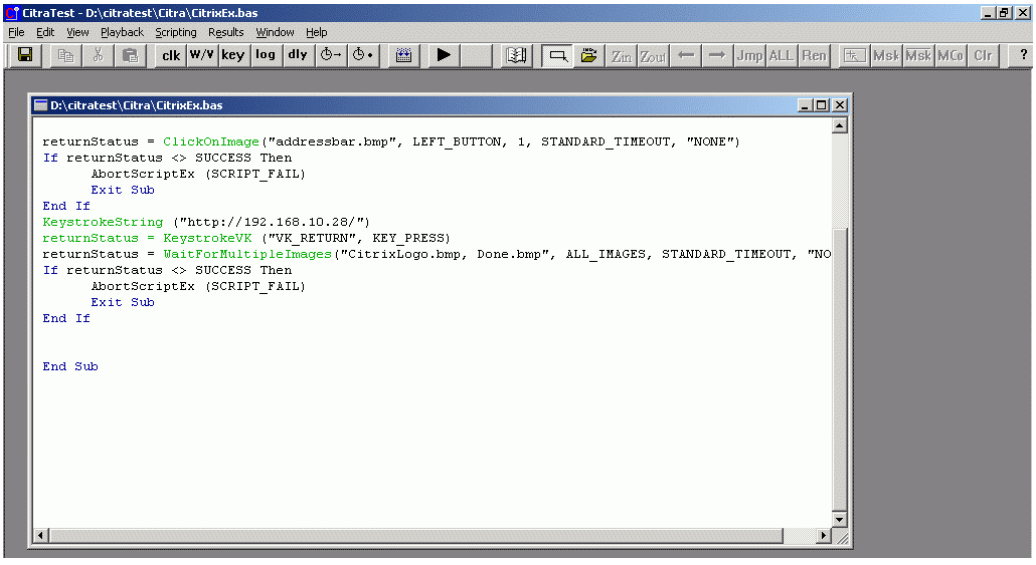

Figure 2.88: Script for waiting for the Citrixlogo.bmp and Done.bmp

10. To calculate the time taken to connect to the Citrix login page, place the cursor on top of the 'wait for' script (see Figure 2.88), and insert a timer named *CitrixLogin* by clicking on the  button on the tool bar of Figure 2.88. Provide the **Timer Name** in Figure 2.89 that appears, and click the **OK** button therein.

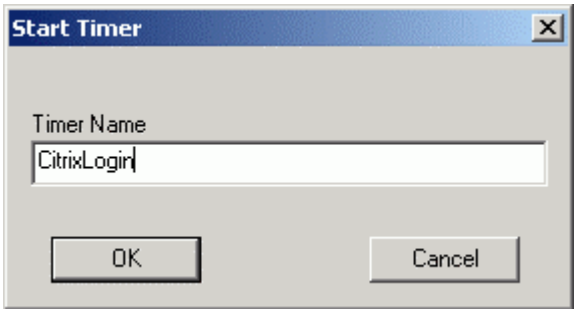

Figure 2.89: Starting the CitrixLogin timer

11. Stop the *CitrixLogin* timer after the 'wait for' script by clicking on the  button on the tool bar of the script window. Select the Timer to be stopped from the **Timer Names** list of Figure 2.90, and then click the **OK** button therein to add the corresponding script code to the VB script

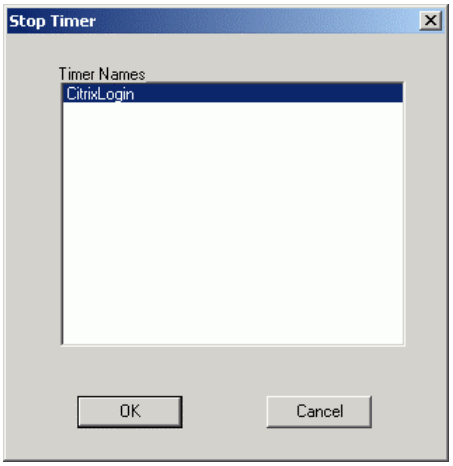

Figure 2.90: Stopping the CitrixLogin timer

12. Next, the script will have to provide the authentication information (username, password, domain) required to login to the Citrix server. Begin by providing the user name *john*. For that, first capture the image of the **Username** text box, and define its click spot, so that the cursor is automatically positioned inside the text box (see Figure 2.91).

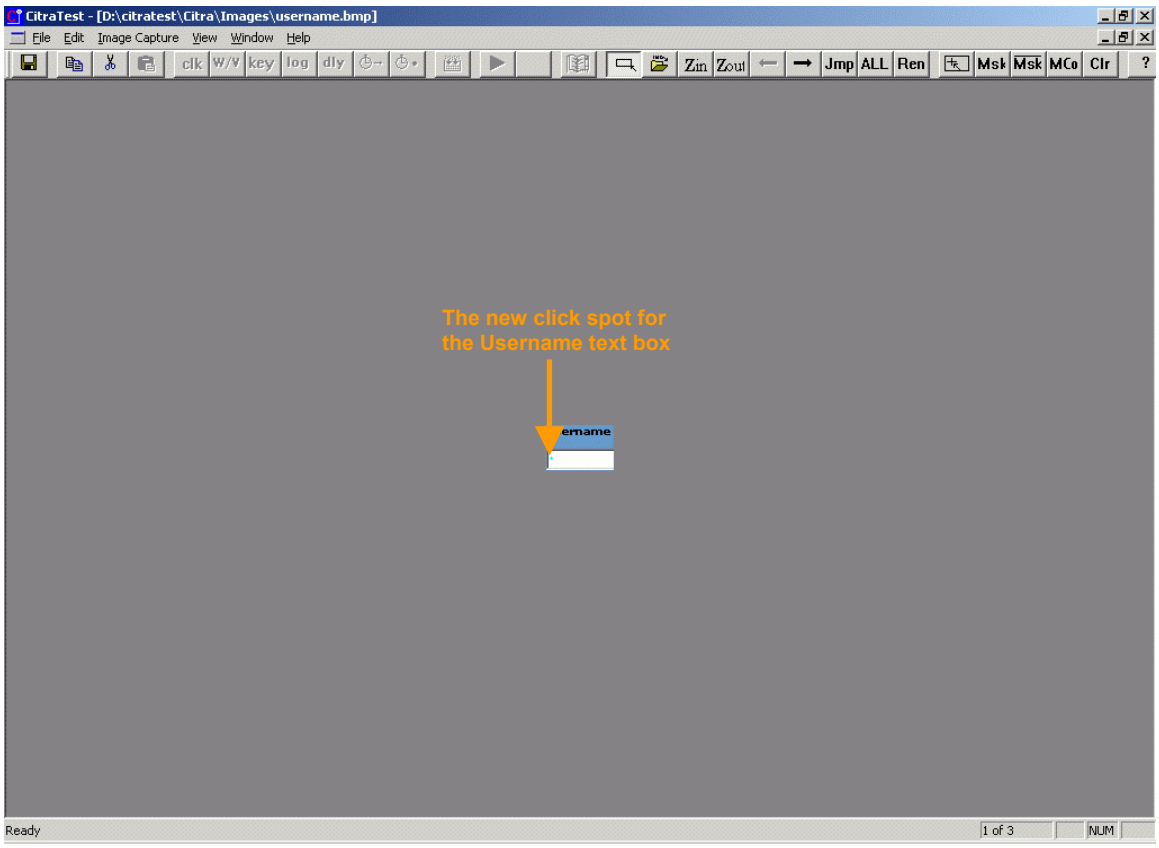

Figure 2.91: Defining the click spot for the username.bmp image

13. Then, associate a click event with the **username.bmp** (see Figure 2.92), and click the **OK** button therein.

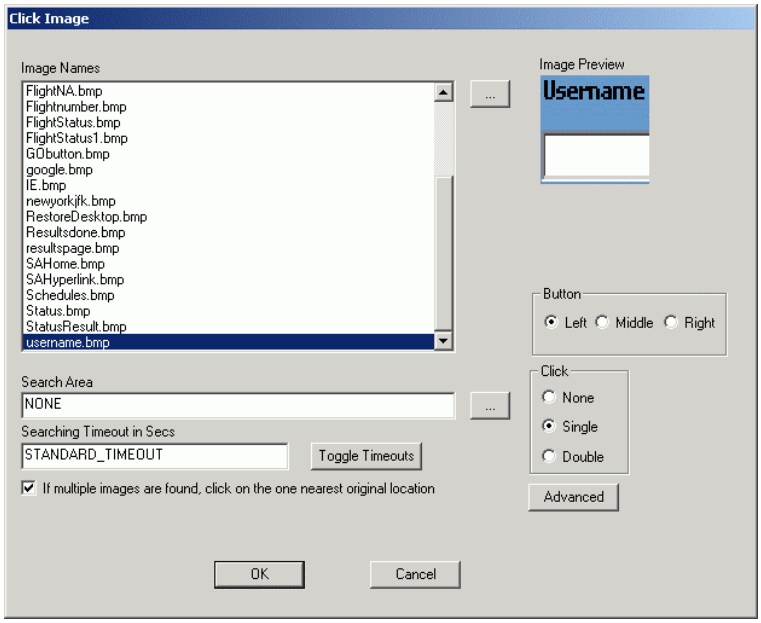

Figure 2.92: Clicking on username.bmp

14. Once you return to the script window, you will find that the script code corresponding to the click event has been appended to it (see Figure 2.93).

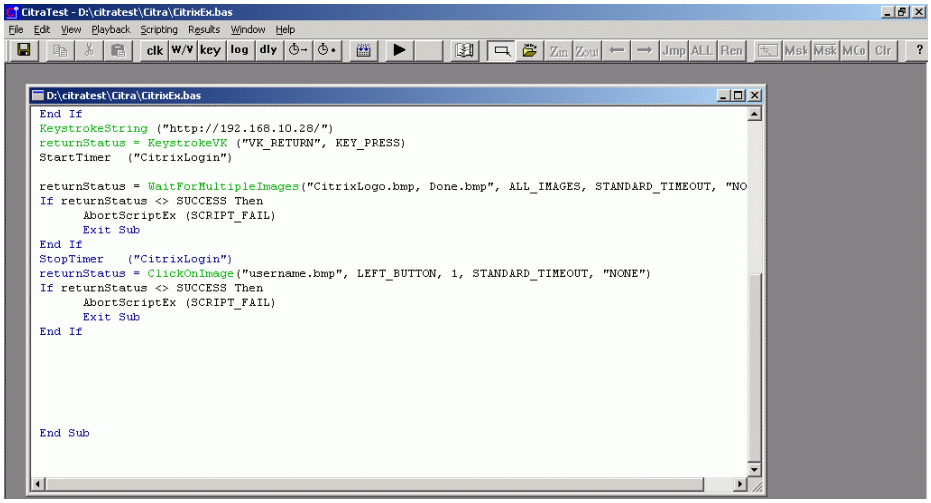

Figure 2.93: Script for clicking on username.bmp

15. Next, type the username *john* in the **Username** text box as depicted by Figure 2.94.

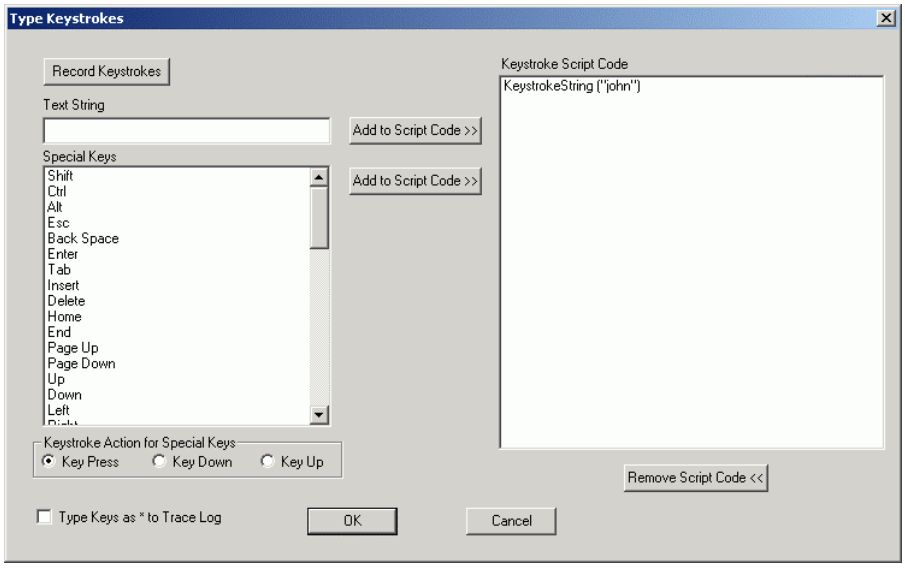

Figure 2.94: Keying in the user name john

16. Now, to navigate to the **Password** field in the login page, the script will have to press the **Tab** key on the key board. Build this capability into the script by following the procedure depicted by Figure 2.95 below.

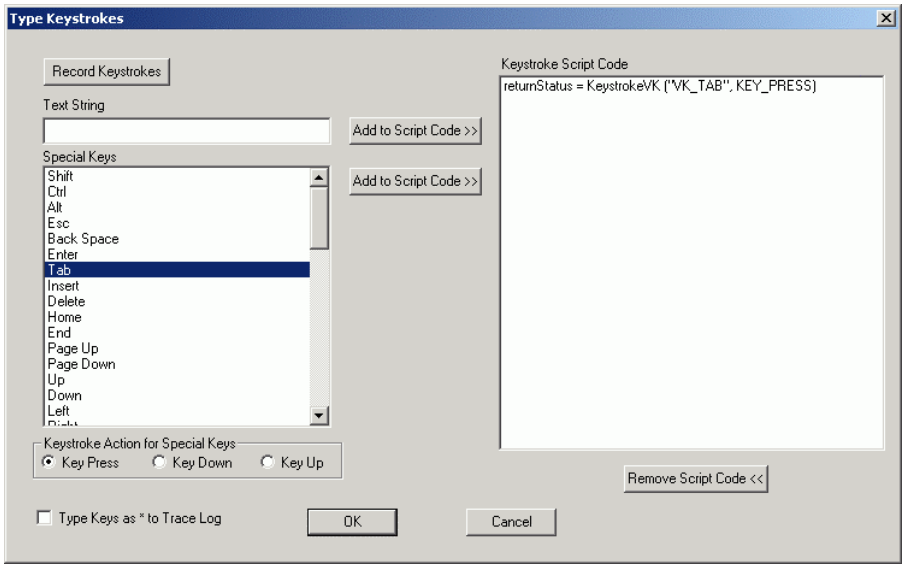

Figure 2.95: Pressing the Tab key to switch to the Password field

17. Once in the **Password** text box, the script needs to type *egurkha* as the password (see Figure 2.96).

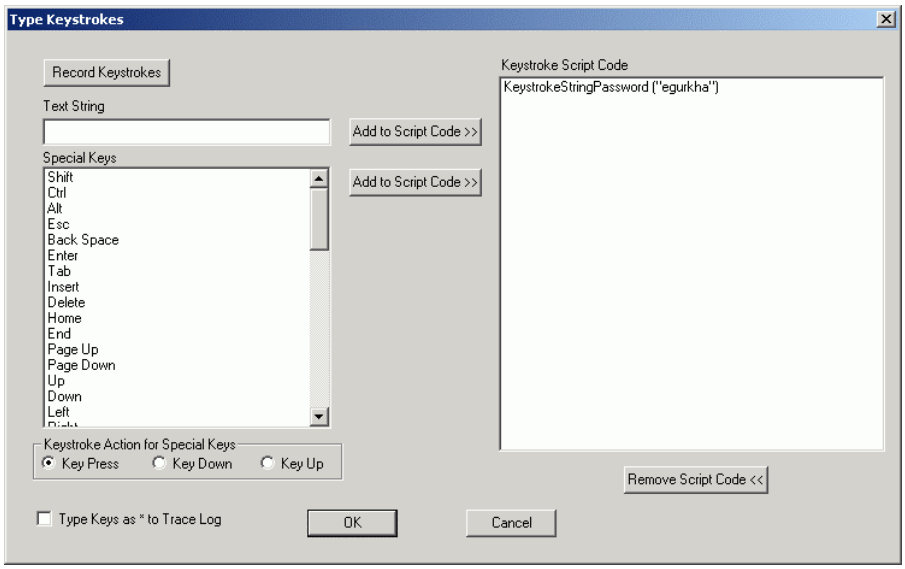

Figure 2.96: Specifying the password, egurkha

18. As *chn* is the **Domain** to which the Citrix user *john* belongs, the script will have to click on the **Tab** key yet again to move to the **Domain** text box, and then, specify *chn* in it. After which, the script will have to click on the **Login** button to login to the Citrix server, 192.168.10.28. To click on the **Login** button, instruct the script to press the **Tab** key from the **Domain** text box, and then press **Enter** (see Figure 2.97).

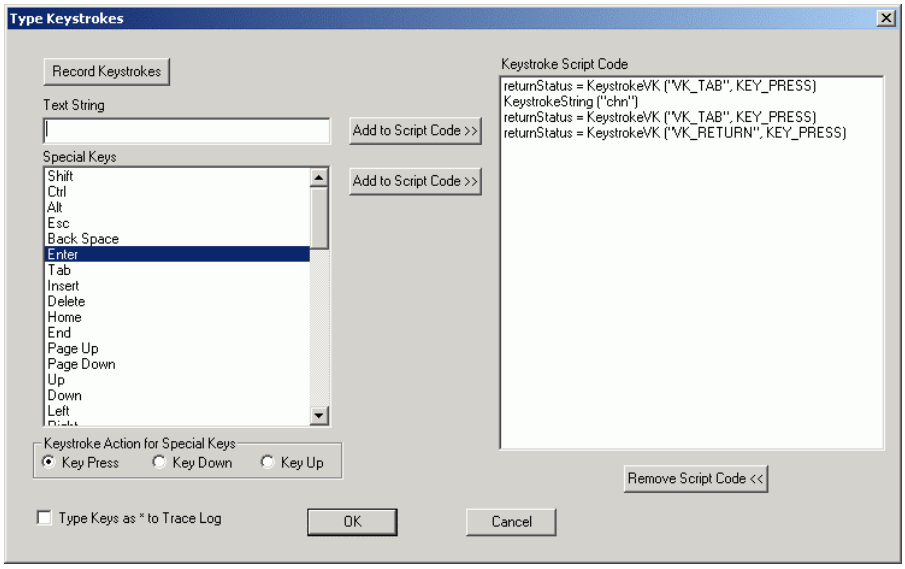

Figure 2.97: Providing the domain name and clicking on the Login button

19. The script window will thus be updated with the login script (see Figure 2.98).

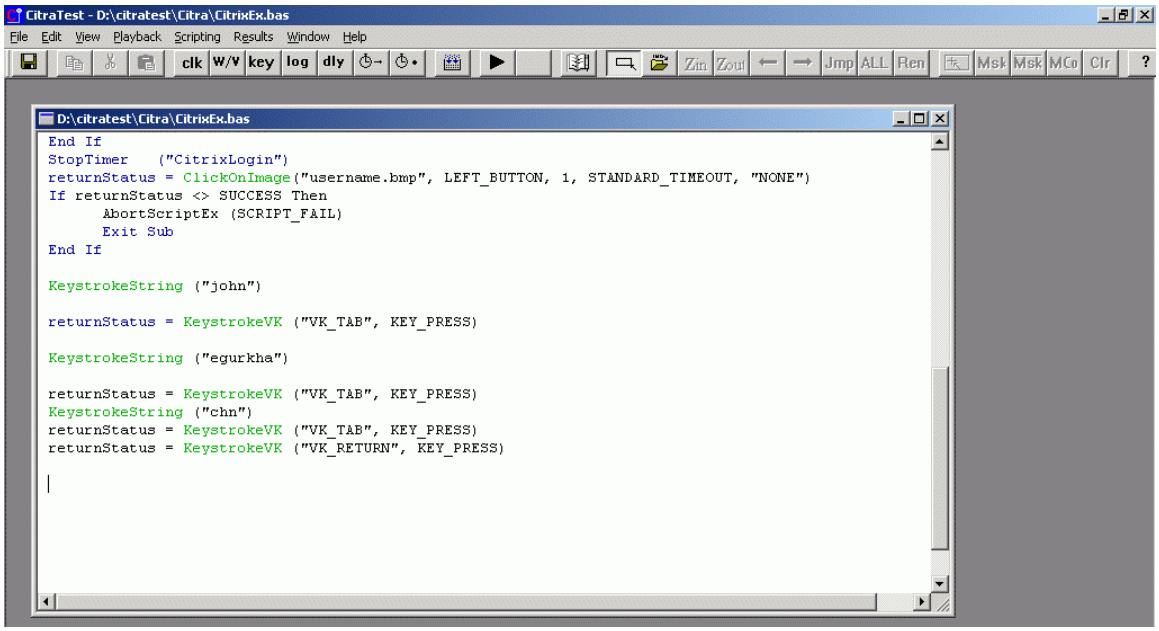

Figure 2.98: Script for logging into the Citrix server, 192.168.10.28

20. If the login is successful, the resultant page will display an **Applications** section, and of course, a **Done** message on the status bar. The script should therefore be instructed to wait for the appearance of the aforesaid to confirm successful login. To achieve this, first, capture a small portion of the title of the **Applications** section (see Figure 2.99).

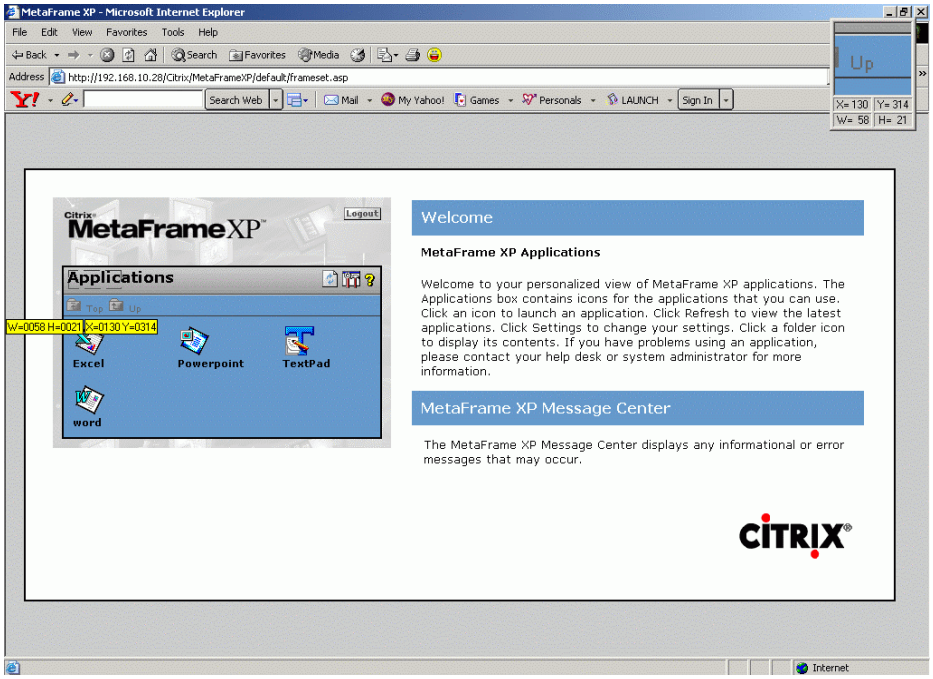

Figure 2.99: Capturing the title of the Applications section

21. Save the image as **Applicationstext.bmp** (see Figure 2.100).

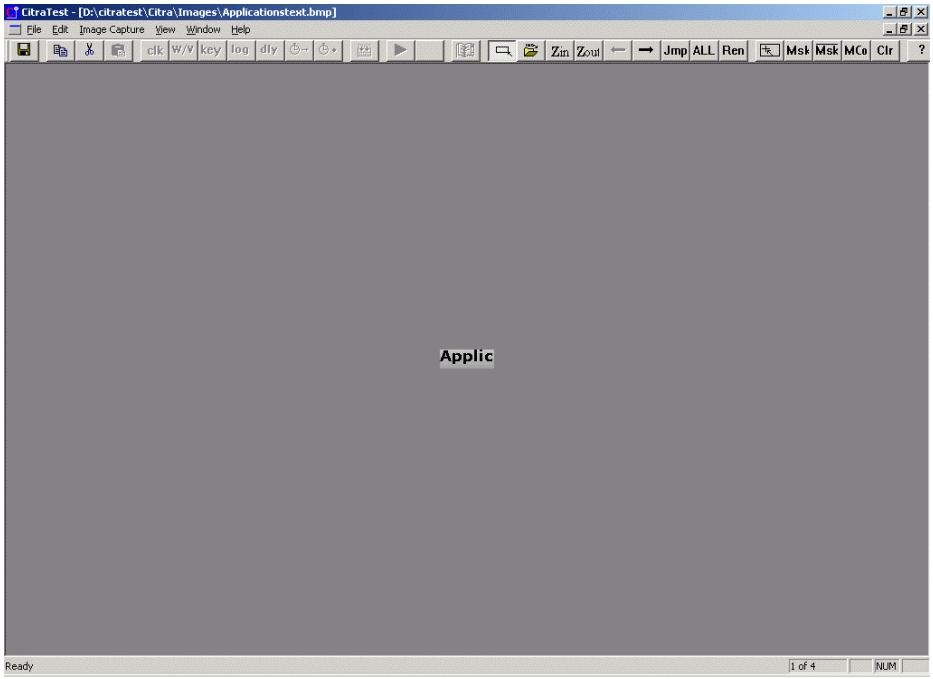

Figure 2.100: Saving the image as Applicationstext.bmp

22. Next, follow the procedure depicted by Figure 2.101 to associate a 'wait for' event with both the **Applicationstext.bmp**, and the already existing **Done.bmp**.

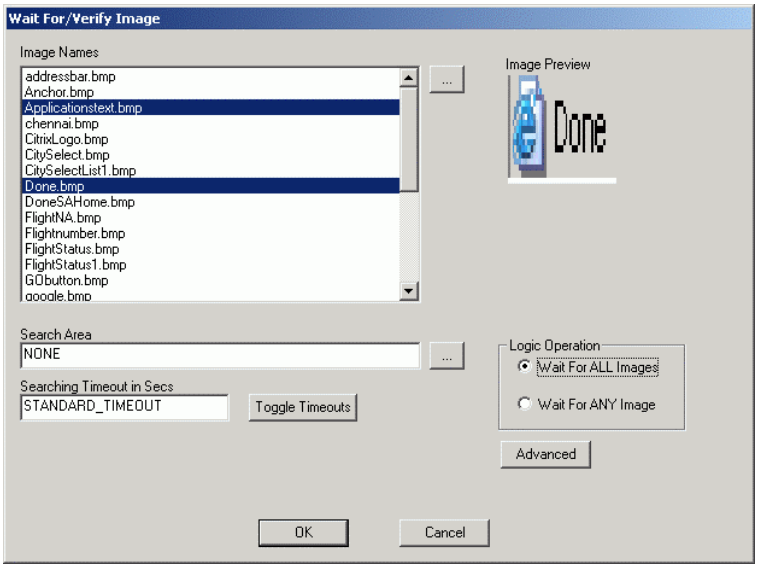

Figure 2.101: Waiting for Applicationstext.bmp and Done.bmp

23. Now, the login time will have to be measured. To ensure this, start a timer named *CitrixConnect* before the 'wait for' script, and stop the timer after the 'wait for' script. The script will be updated accordingly (see Figure 2.102).

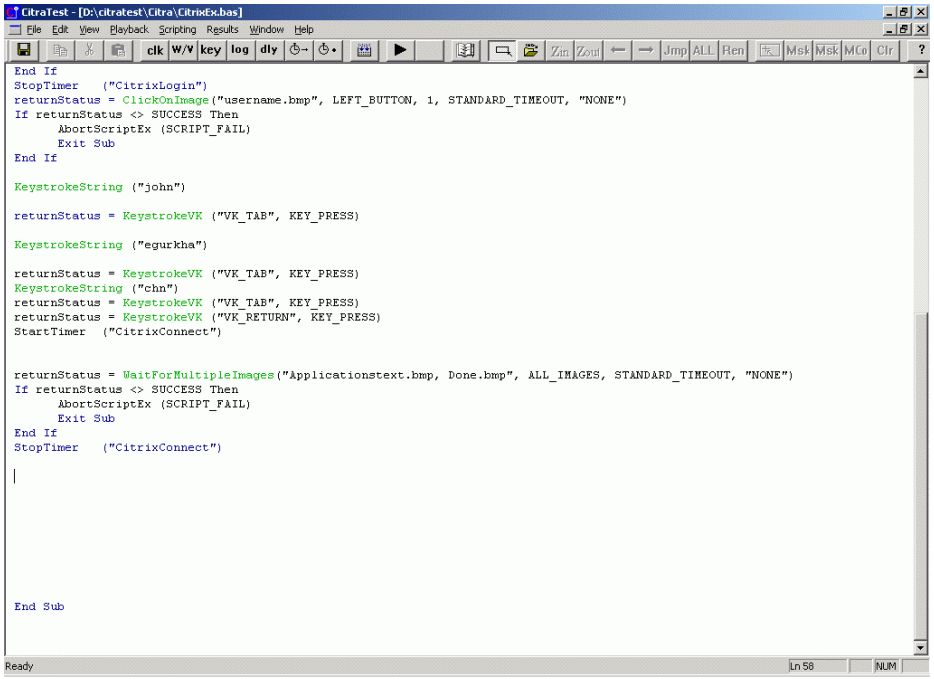

Figure 2.102: Script for calculating the login time

24. Now that we have gained access to the Citrix server, 192.168.10.28, let us proceed to open the **Textpad** application published on it. As the first step towards this, capture an image of the **Textpad** icon as shown by Figure 2.103.

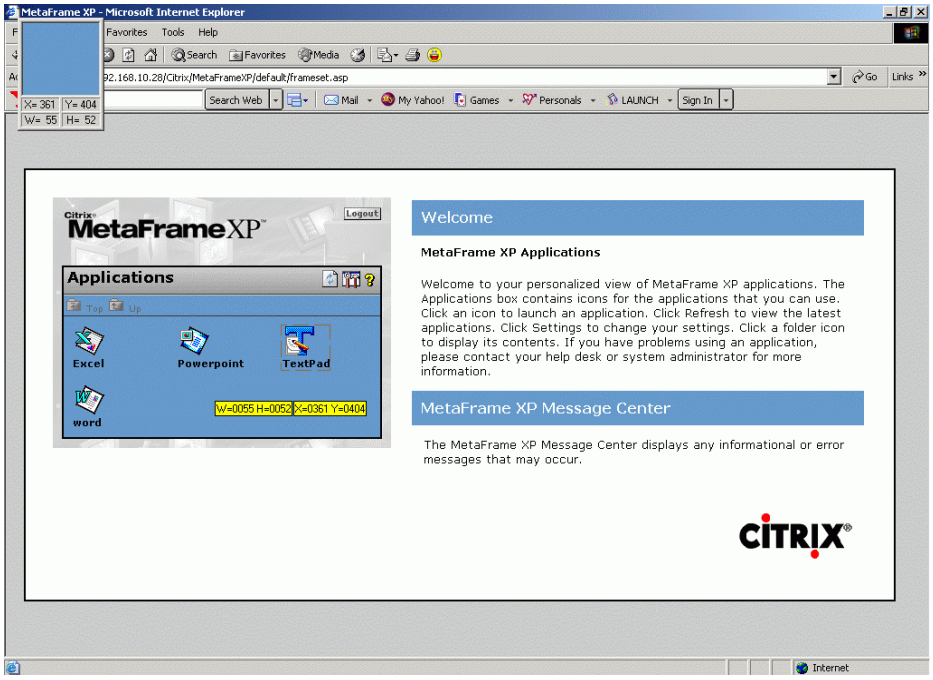

Figure 2.103: Capturing as image of the Textpad icon

25. Save the image as **Textpadicon.bmp** (see Figure 2.104).

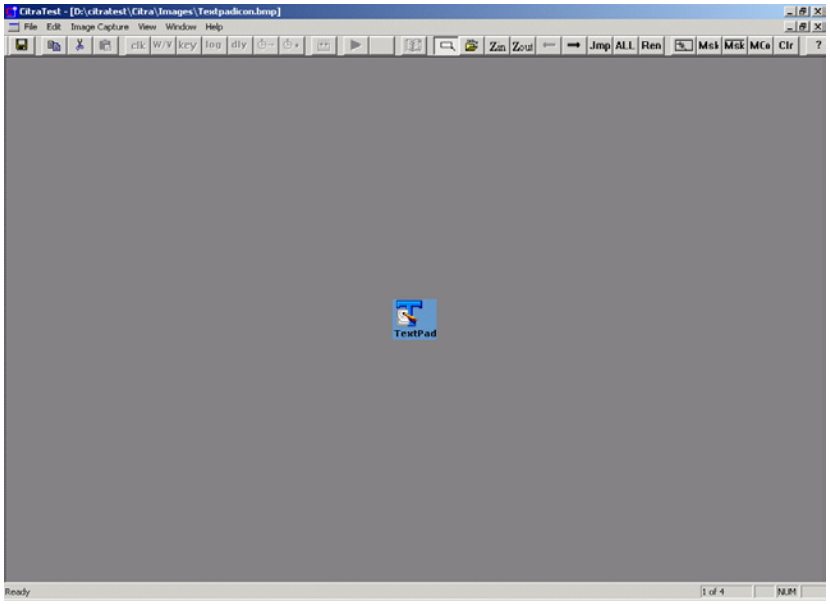

Figure 2.104: Saving the Textpadicon.bmp

26. The next step is to click on the **Textpad** icon. To achieve this, first, click on the **clk** button on the script window. Then, proceed as shown by Figure 2.105.

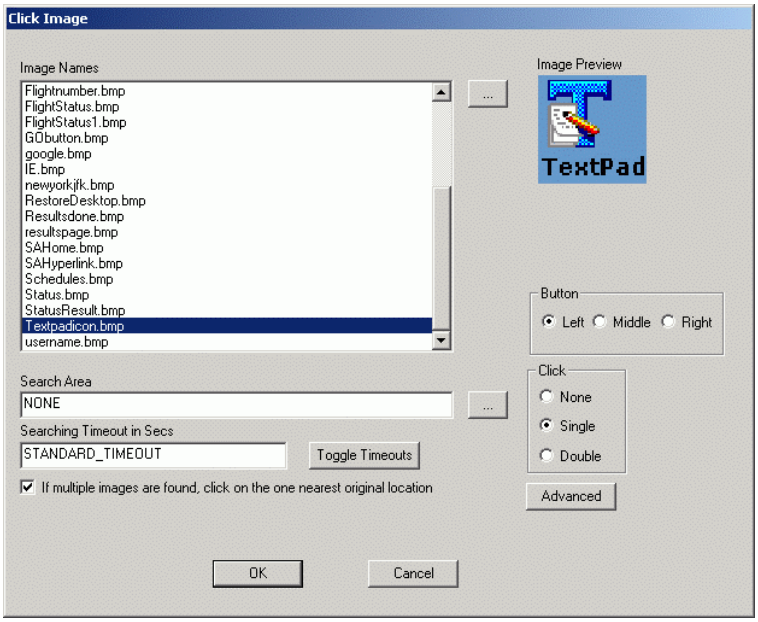

Figure 2.105: Clicking on the Textpadicon.bmp

27. Figure 2.106 displays the script code that corresponds to the association of a click event with the **Textpadicon.bmp**.

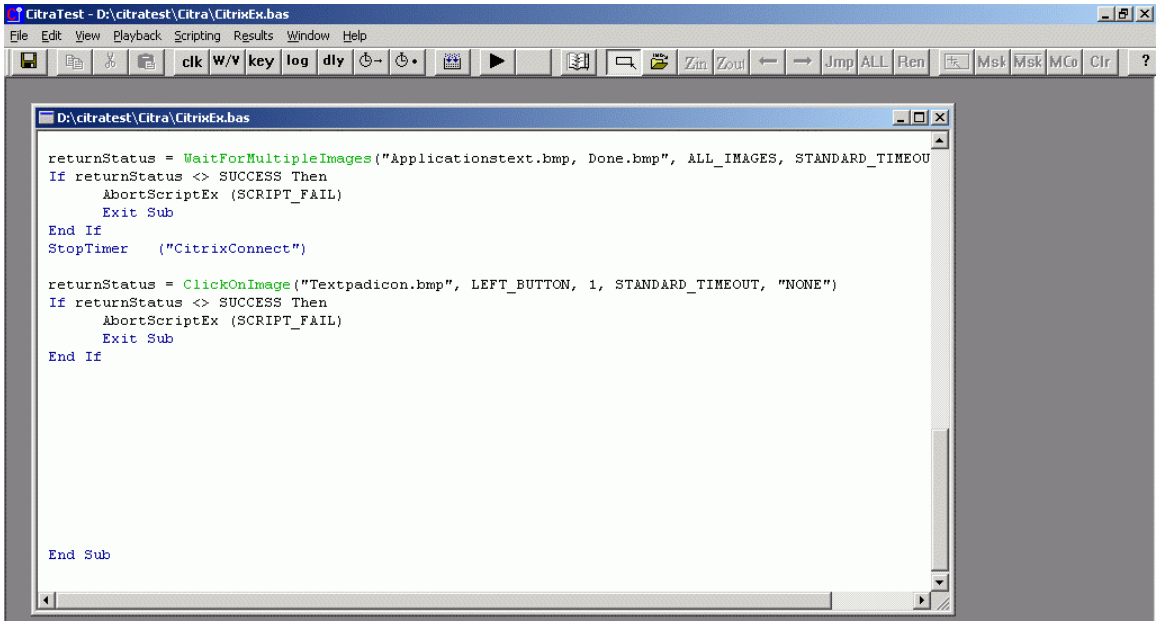

Figure 2.106: Script for clicking on the Textpadicon.bmp

28. Next, the script will have to wait until the **Textpad** application fully opens. A clear indicator to this is the appearance of the title bar text, *TextPad – [Document 1]*. Therefore, open the **Textpad** application, and capture an image of its title bar text (see Figure 2.107).

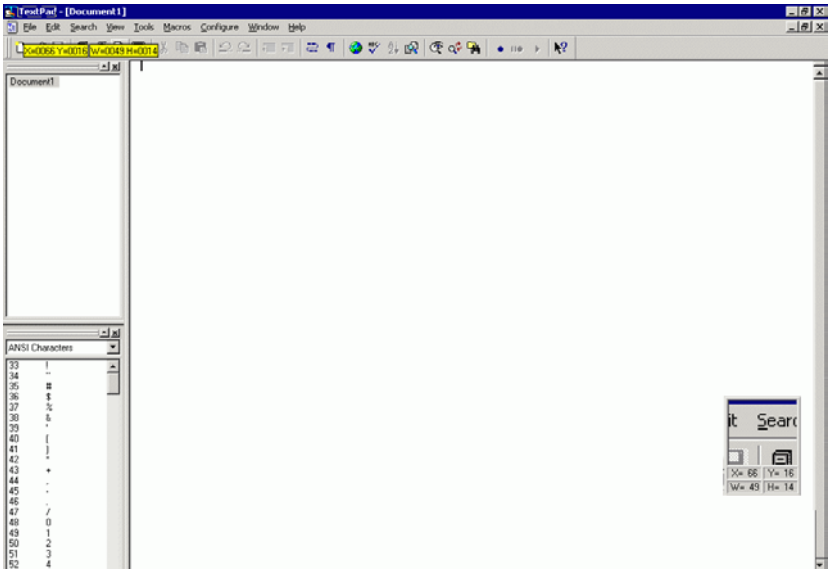

Figure 2.107: Capturing an image of the text on Textpad's title bar

29. Save the image as **TextpadTitle.bmp**.

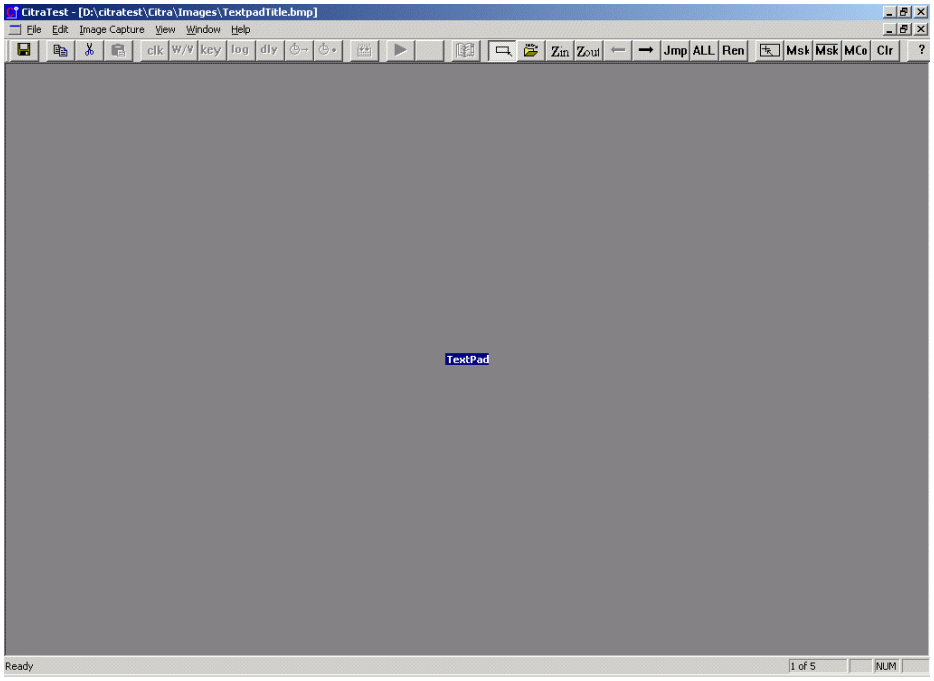

Figure 2.108: Saving the image as TextpadTitle.bmp

30. Then, to make sure that the script waits for the **TextbarTitle.bmp** image to appear, click on the **W / V** button on the script window's tool bar, and do as indicated by Figure 2.109 below.

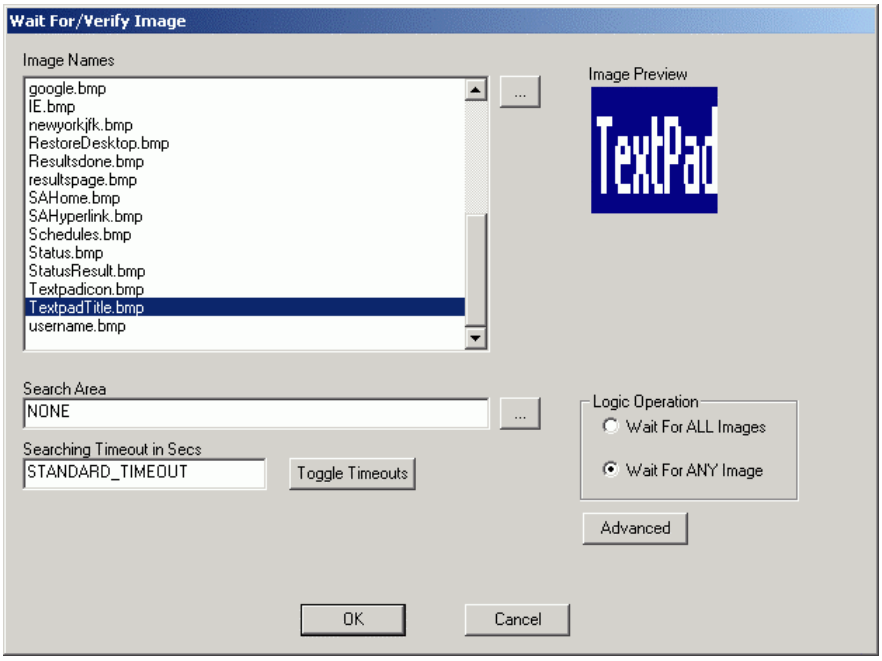

Figure 2.109: Waiting for TextbarTitle.bmp

31. Also, track the time taken for Textpad to open, by starting a timer named *TextpadOpen* before the 'wait for' script, and then stopping it after the 'wait for' script (see Figure 2.110).

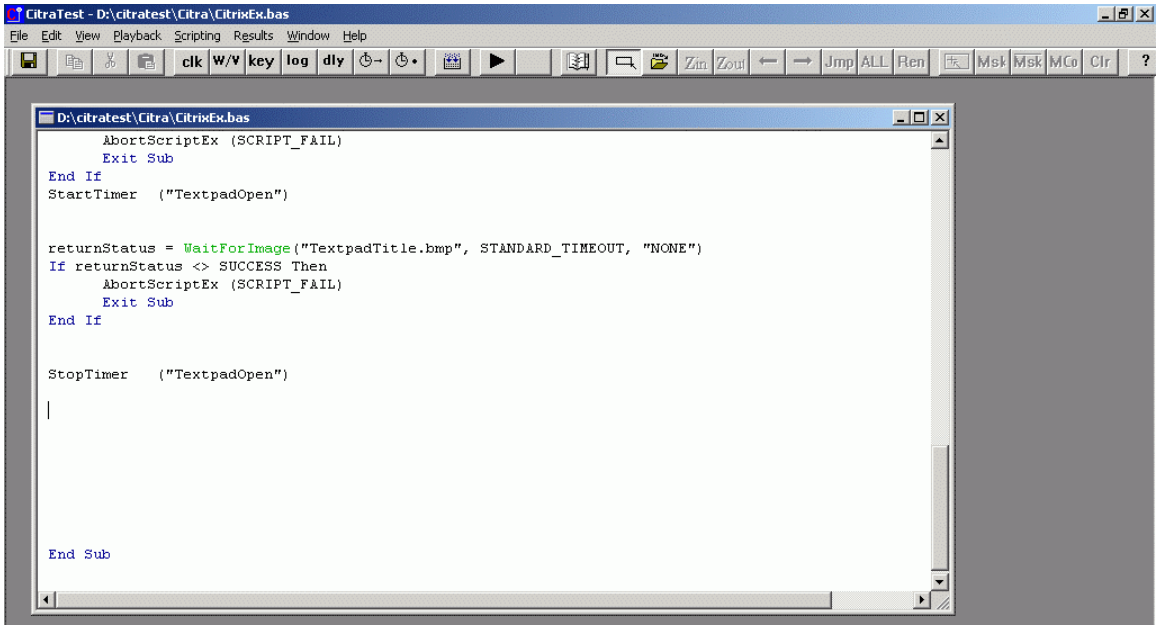

Figure 2.110: Calculating the time taken for opening Textpad

32. Let us now proceed to type a few words in Textpad. To instruct the script to do the same, click on the **key** button in Figure 2.110 and proceed in the same manner depicted by Figure 2.111.

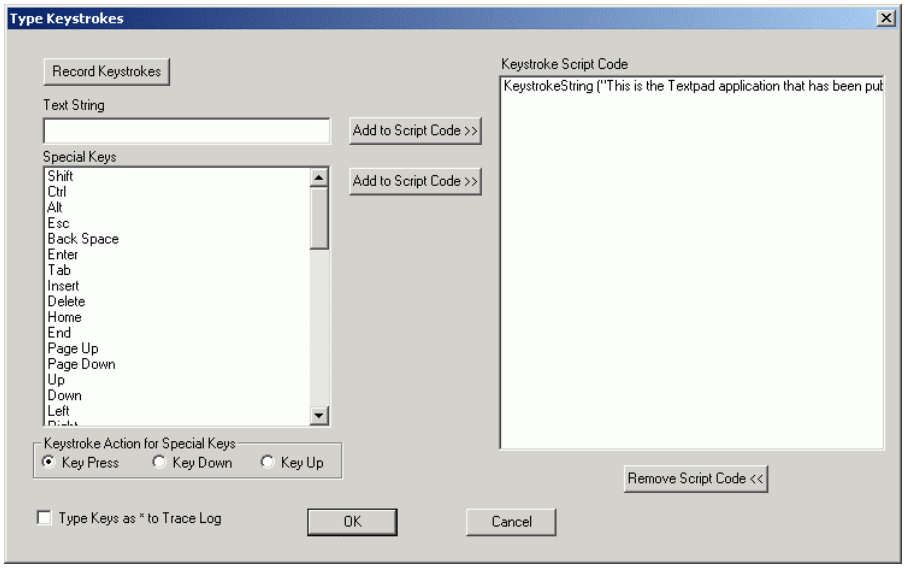

Figure 2.111: Typing a line in text pad

33. The script is accordingly updated (see Figure 2.112).

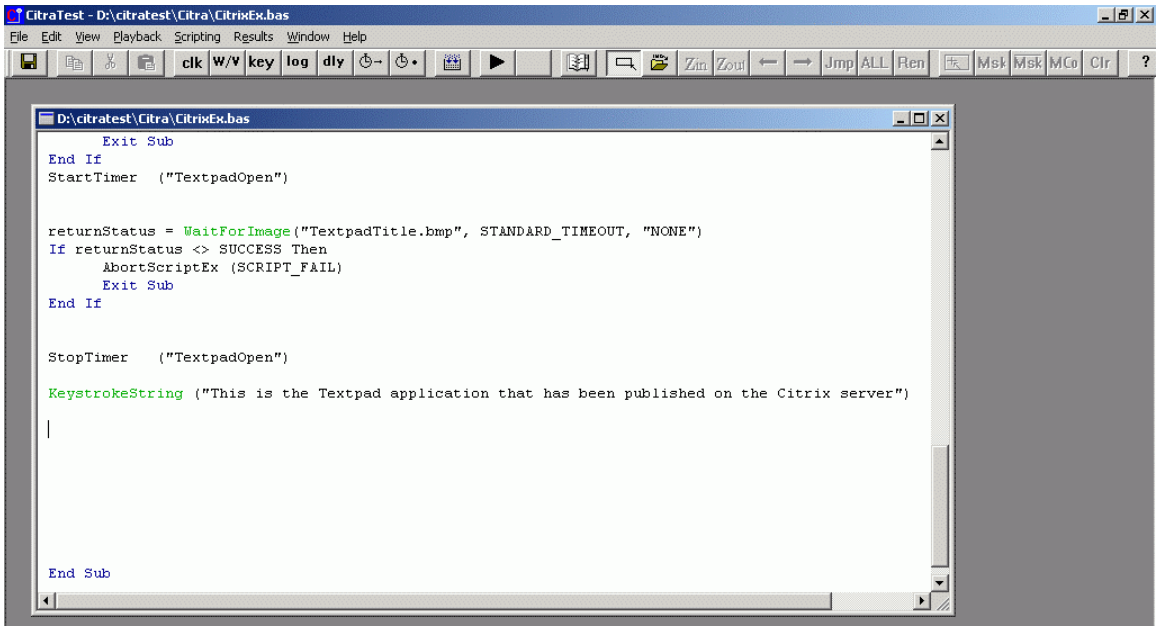

Figure 2.112: Script for typing a line of text on Textpad

34. Finally, the script will have to close the Textpad application without saving the changes made to it. The simplest way to initiate application closure is to press the "Alt+F4" key combination. To achieve this, once again, click on the **key** button in Figure 2.112. From the **Special Keys** list of Figure 2.113, select **Alt**. Since the **F4** key is to be pressed while

holding down the **Alt** key, choose the **Key Down** option from the **Keystroke Action for Special Keys** section. Then, click on the **Add to Script Code >>** button. Next, select **F4** from the **Special Keys** list, choose the **Key Press** option, and then click on the **Add to Script Code >>** (see Figure 2.114).

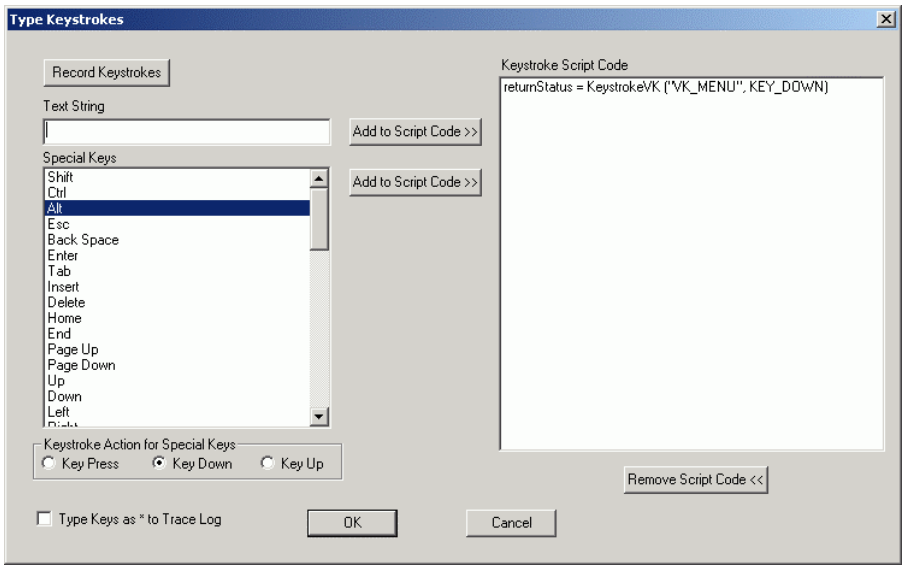

Figure 2.113: Pressing down the Alt key

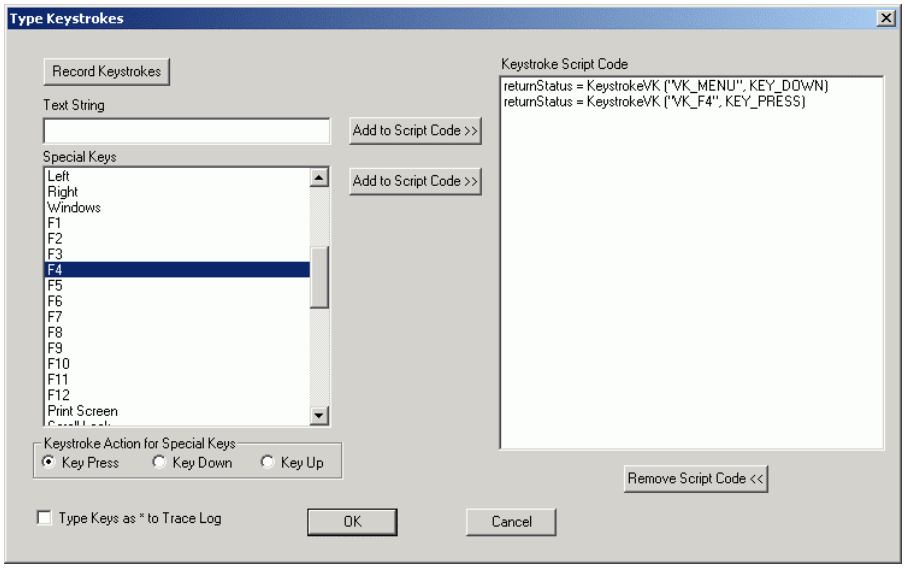

Figure 2.114: Pressing the F4 key

35. As soon as "Alt+F4" is pressed, a message box requesting the user's confirmation to save the changes made to the Textpad application will appear. The script will have to click on the **No** button here. By default, control will rest on the **Yes** button in the message box. To move to the **No** button, the **Tab** key is to be pressed. To click on the **No** button, the **Enter** key is to be pressed. Before performing either of these steps, the **Alt** key that was "pressed down" earlier, should be released. To ensure this, click on the **key** button on the

script window's tool bar, select **Alt** from the **Special Keys** list of Figure 2.115, select the **Key Up** option from the **Keystroke Action for Special Keys** section, and then, click the **Add to Script Code >>** button.

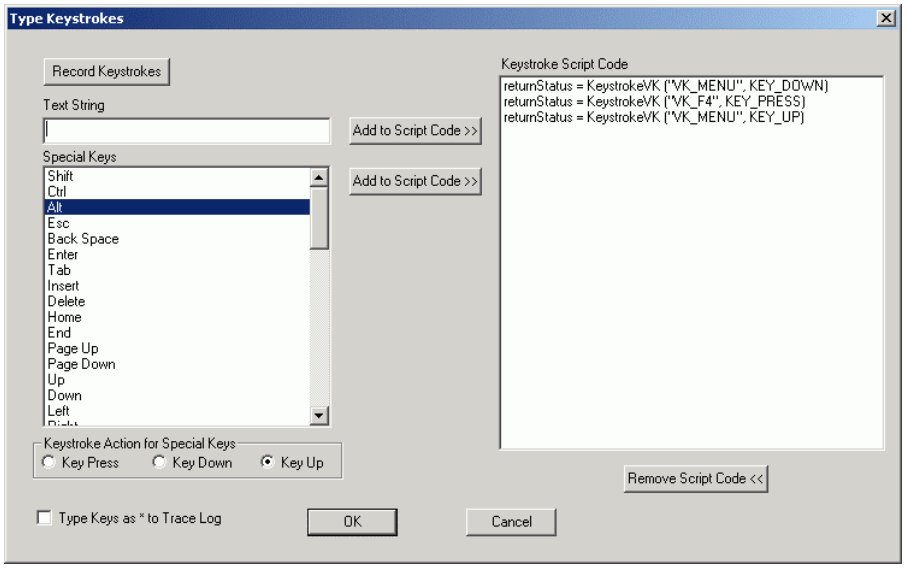

Figure 2.115: Releasing the Alt key

36. Finally, click the **OK** button in Figure 2.115 to return to the script window. After verifying the changes that were effected on the script, click on the **key** button on the tool bar once again. This time select the **Tab** option from the **Special Keys** list and click on the **Add to Script Code >>** button adjacent to it. Similarly, add the **Enter** option to the **Keystroke Script Code** list (see Figure 2.116). Then, click the **OK** button to return to the script window and view the complete script (see Figure 2.117).

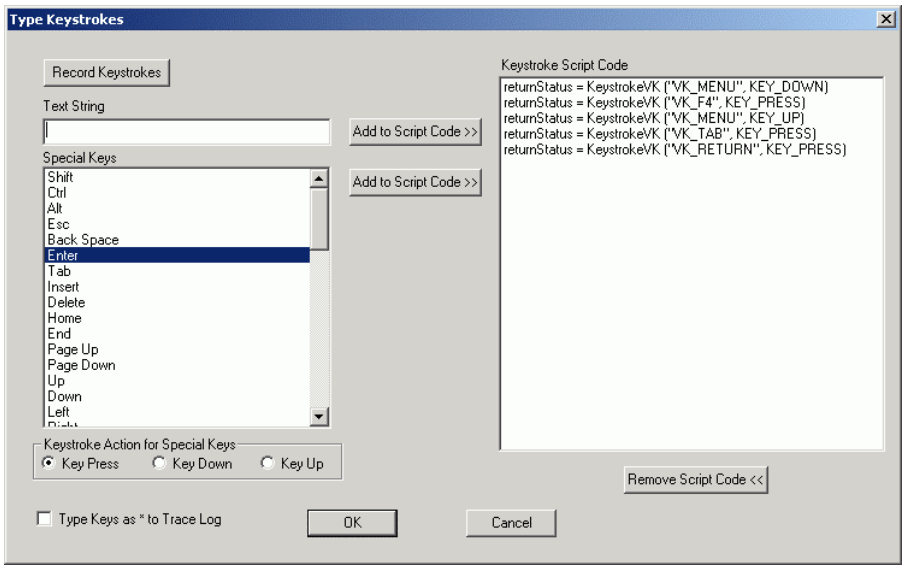

Figure 2.116: Pressing the Tab key and then the Enter key to click on the No button in the message box

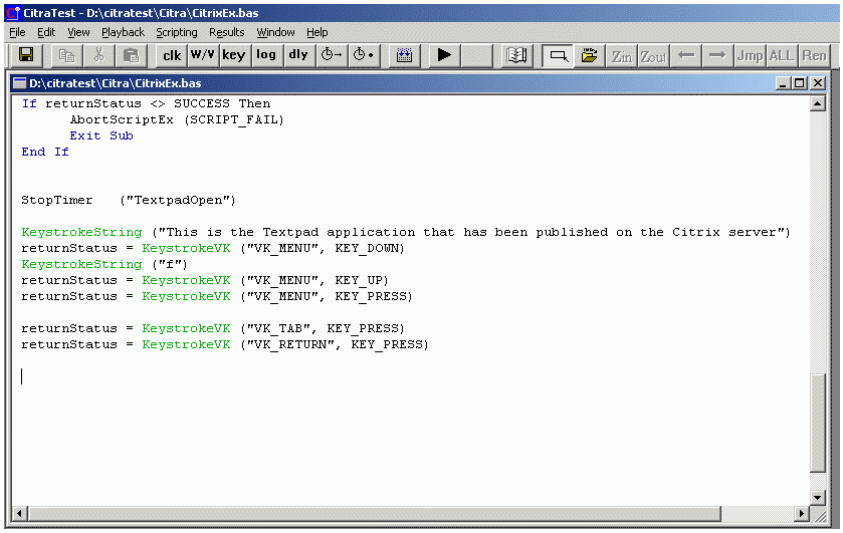

Figure 2.117: The complete script

37. With that, request emulation is complete. Now, compile the script by clicking the button on the tool bar of Figure 2.117. Once compilation becomes successful, run the script by clicking on the button.

2.3.2 Configuring the eG manager to Work with CitraTest

Since the component to be monitored in our second example is the Citrix server 192.168.10.28:1494, add this server as an *Emulated C*lient. The procedure for adding an *Emulated Client* has already been dealt with in Example 1. Therefore, let us proceed to configure the CitraTest for the Citrix server 192.168.10.28:1494.

To achieve this, do the following:

1. After adding the *Emulated Client* 192.168.10.28:1494, attempt to sign out of the eG administrative interface.
2. From the list of unconfigured tests that appears, click on **CitraClientEmulation** test.
3. In the test configuration page that appears, specify the following (see Figure 2.118):

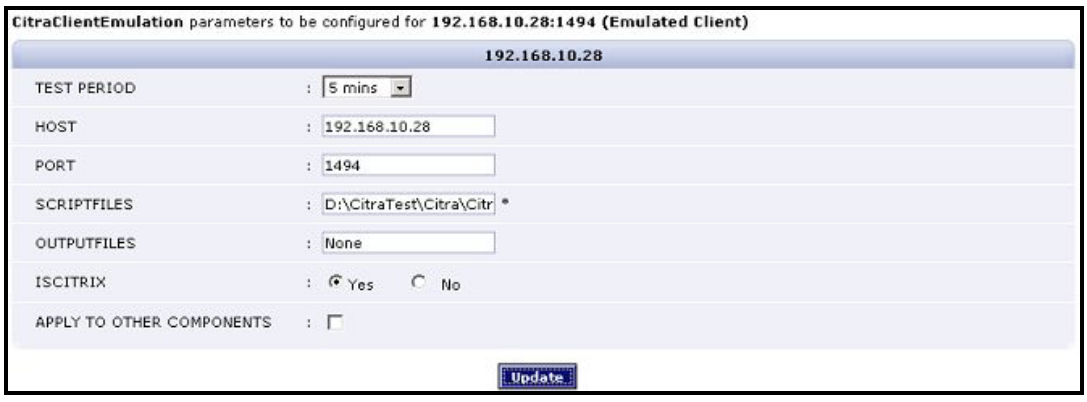

Figure 2.118: Configuring CitraClientEmulation test for 192.168.10.28:1494

a. **TEST PERIOD** – How often should the test be executed

b. **HOST** – The host on which the test will run. In our example, the test will attempt to extract measures from the host, 192.168.10.28.

c. **PORT** – The port at which the specified **HOST** listens. In our example, this is port 1494 – the port at which the Citrix client listens.

d. **SCRIPTFILES** – Specify the full path to the script file that is to be played back for emulating a request to, and extracting metrics from the monitored application. Multiple script files can be provided as a comma-separated list, but all script files should monitor the same application only. In our example, the path to the **CitrixEx** script has to be specified here.

> **Note:**
>
> If the script file resides on another host, then ensure that the location of the script file is mapped to any drive on the measurement host.

e. **OUTPUTFILES** – Enter the full path to the output file that contains the metrics extracted by the specified script file. Here again, multiple output files can be provided as a comma-separated list, but only if multiple script files are also provided.

> **Note:**
>
> - If **None** is specified here, then the eG system will collect statistics from the default output files associated with each of the specified script files. The default output files will be present in the same location as the respective script files, and will have the same name as the script files. In our example, the value of the **OUTPUTFILES** parameter can remain as **None**.
> - While specifying multiple output files, ensure that they are provided in the same order as their corresponding script files in the **SCRIPTFILES** text box.
> - If the **SCRIPTFILES** parameter consists of multiple entries and the **OUTPUTFILES** parameter consists of only one, then eG will automatically associate the first script file entry in the **SCRIPTFILES** box with the **OUTPUTFILES** entry. Measures pertaining to the other script files will therefore not be displayed in the eG monitor interface.

f. **ISCITRIX** – If the specified script emulates a request to a Citrix client then specify **Yes** here. If not, specify **No**. Our example does attempt to extract measures from a Citrix client. Therefore, provide **Yes** here.

4. Finally, click on the **Update** button in Figure 2.118, and then, signout of the administrative interface.

2.3.3 Starting the External Agent

Refer to the *Section 2.4* for details on what is to be done before starting the external agent associated with an *Emulated Client* component.

2.3.4 Viewing the Measures

To view the measures reported by the **CitraClientEmulation** test, do the following:

1. Login to the eG monitor interface as *supermonitor* with password *supermonitor*.
2. From the **Components** menu, select the **Servers** option. Once the **COMPONENT LIST** page appears, select the *Emulated Client* option from the **Type** list therein, so as to view the current state of all components of type *Emulated Client*. Then, click the **Submit** button (see Figure 2.79).

Figure 2.119: The COMPONENT LIST page

3. The *Emulated Client* component that we had configured previously will then be listed (see Figure 2.79). Click on the component to view its layer model, tests, and measurements (see Figure 2.80).

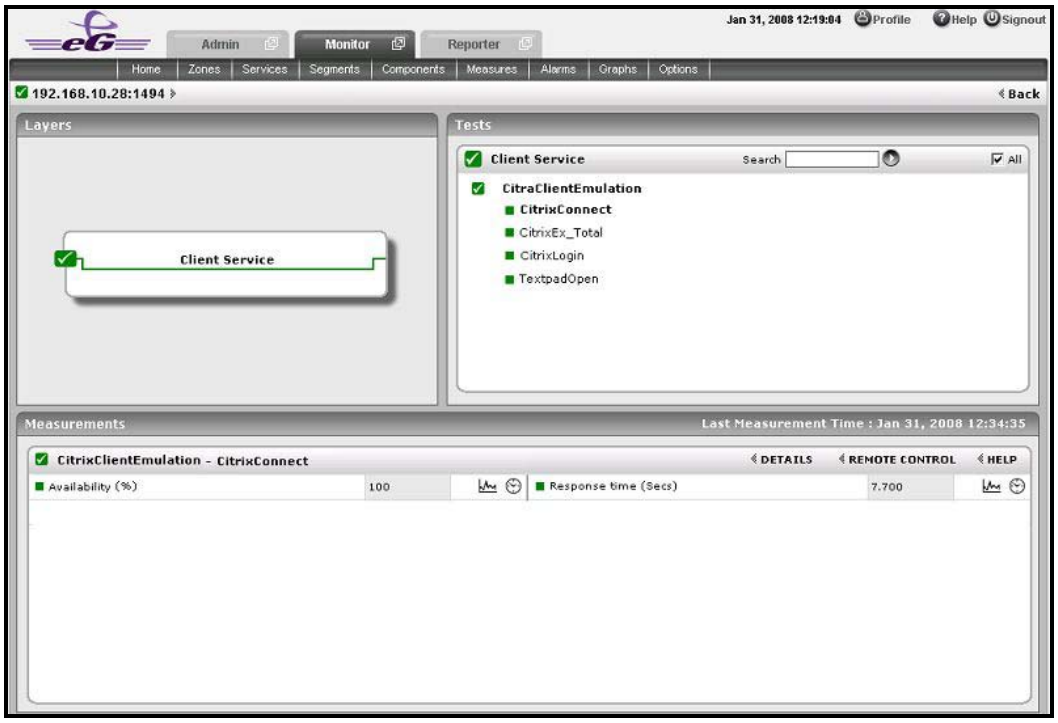

Figure 2.120: Viewing the layer model, tests, and measurements of the Emulated Client

4. A timer maps to a step in a multi-step service interaction. eG monitors steps using timers. Figure 2.120 displays the **Availability** and **Response time** of the *CitrixConnect* step built into the **CitrixEx** script. This step calculates the time taken by a Citrix user to login to the Citrix application, and reports this time as the **Response time**. Also, note that all the steps that have been included in the **CitrixEx** script appear as descriptors of the test (see Figure 2.120).

5. Besides the steps that we had explicitly monitored using the script, Figure 2.120 also displays an additional step named, *CitrixEx_Total*. This step, which is internally generated by the CitraTest tool, reports the total time taken for script execution as the **Response_time**. Figure 2.121 reveals the response time value returned by the *CitrixEx_Total* step. This is the total time taken to login to the Citrix server, 192.168.10.28:1494, open the Textpad application published on it, add a line of text to the Textpad, and close the application without saving the changes.

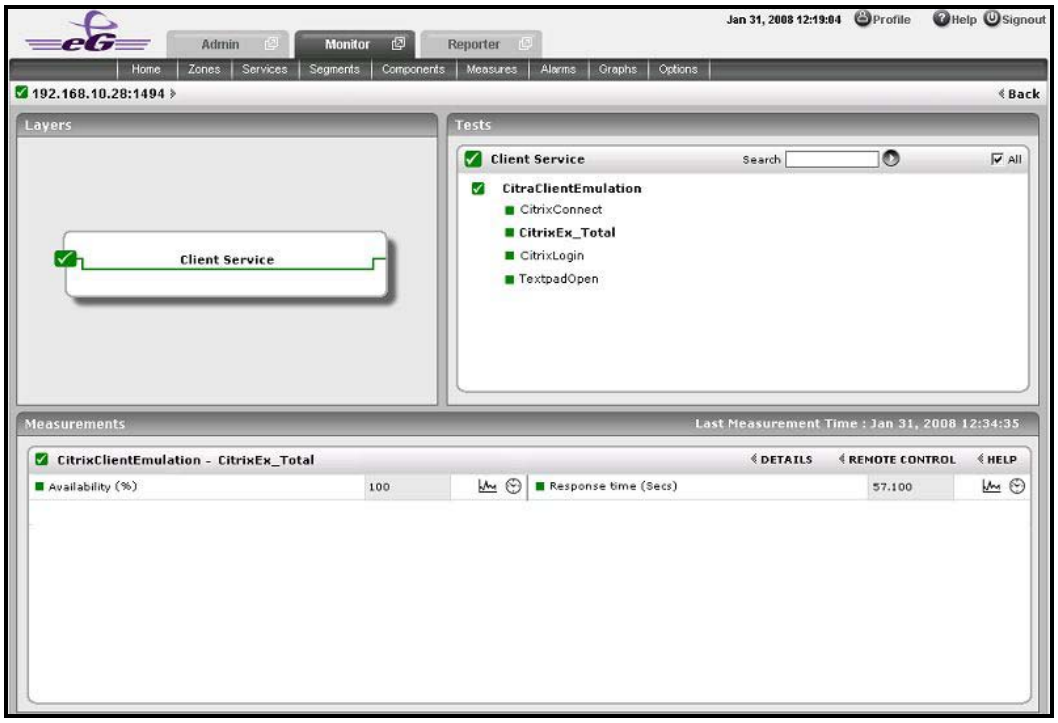

Figure 2.121: Viewing the measures reported by the CitrixEx_Total timer

6. To verify the correctness of the reported values, switch to the system hosting the CitraTest Development environment, and open the **CitrixEx** script in the script window. Then, open the Timer log by following the menu sequence Results -> View Timer Log.

7. The **CitrixEx.timer.log** file of Figure 2.122 reveals the start time, end time, and the duration of every activity performed by the **CitrixEx** script. According to Figure 2.122, the duration of the timer *CitrixConnect* is *7.7 seconds*. Note that the eG monitor interface of Figure 2.120 reveals the same value.

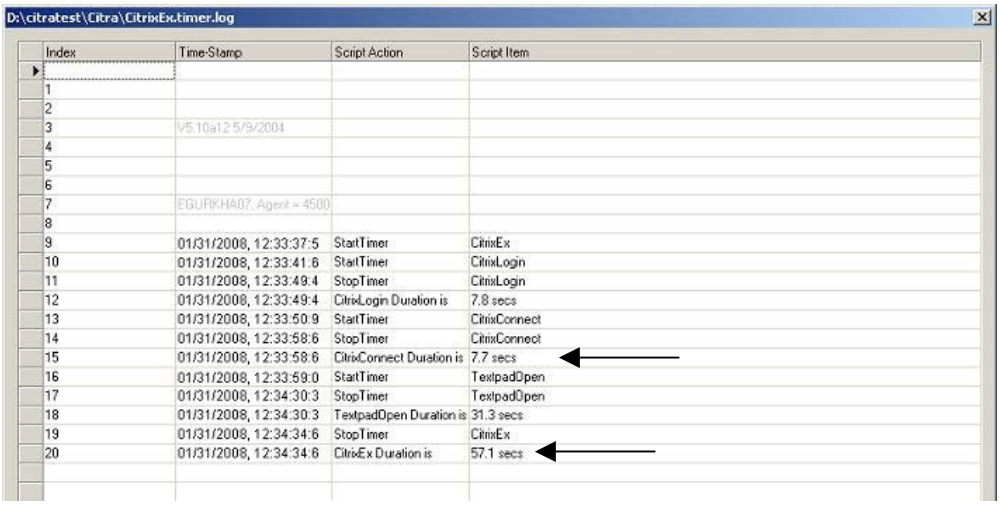

Figure 2.122: The timer log file of the CitrixEx script

8. Similarly, note the value displayed against *CitrixEx Duration* in Figure 2.122. This is the same as the value of the **Response time** measure of the *CitrixEx_Total* timer.

2.4 Tips for Effective Client Emulation

2.4.1 Handling Random Popups and Dialogs

During script playback, some messages popup suddenly causing script execution to terminate. For example, error messages, virus scans, etc. In order to ensure that script execution is not disturbed, these message popups need to be handled as **Exceptions**.

Typically, it would be difficult for a developer to determine what these messages are and when they popup. Under such circumstances, adopt the methodology described below to handle the popups:

1. From the CitraTest script window, select **Playback Options** from the **Playback** menu. From the **Playback Options** dialog box that appears, select the **Screen Event Handling** tab (see Figure 2.123).

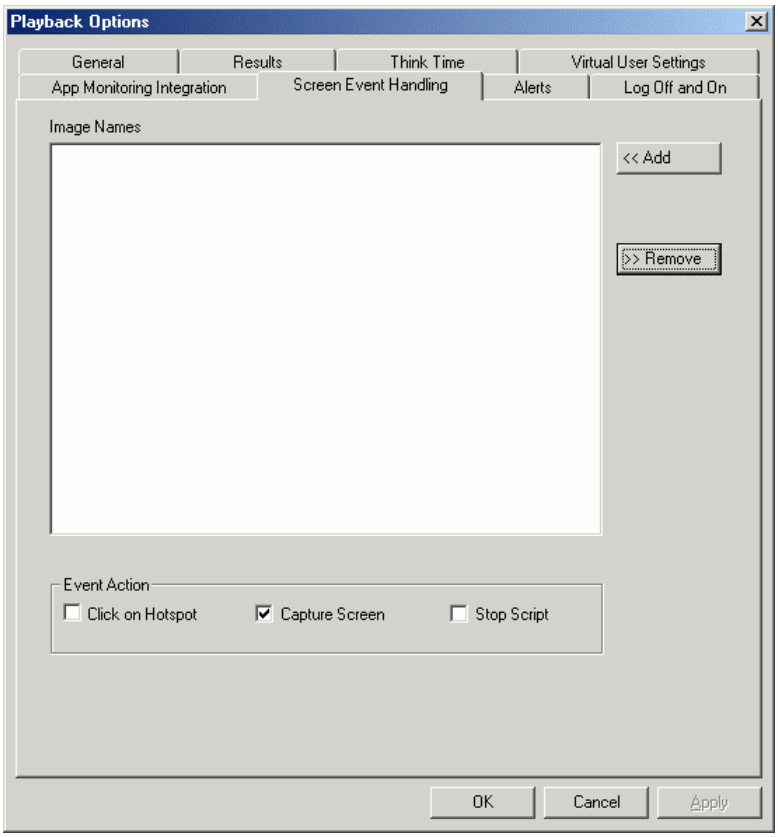

Figure 2.123: The Screen Event Handling tab

2. Select the **Capture Screen** checkbox in Figure 2.123, click the **Apply** button therein, and finally, the **OK** button.

3. Enabling screen capturing ensures that all messages that randomly popup during script execution are automatically captured by CitraTest. Every screen so captured by CitraTest will be named as *PopupScreen1, PopupScreen2, . . .* etc. By default, the screens will be stored in the **Images** directory associated with the project under consideration.

4. The next step is to instruct CitraTest on how to handle the captured popups. For that, open the captured image using the CitraTest script window (see Figure 2.124).

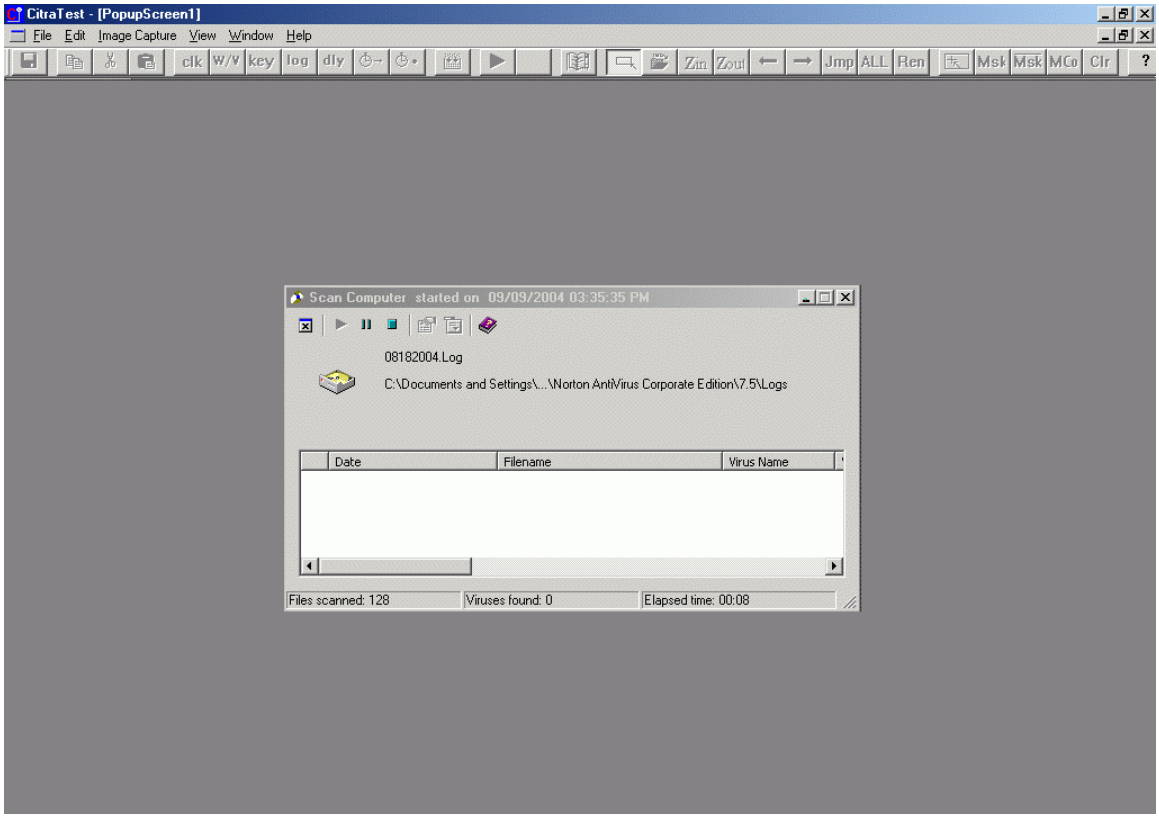

Figure 2.124: Opening a captured popup

5. In the example depicted by Figure 2.124, an automatically captured popup named *PopupScreen1*has been opened. *PopupScreen1*, in this example, is the image of a virus scan's progress tracker. In order to ensure that the virus scan does not affect script execution, CitraTest has to minimize the progress tracker. To achieve this, the **Minimize** button of the tracker has to be set as the click spot of *PopupScreen1*. To do this, click on the button in the tool bar of Figure 2.124, and then click on in *PopupScreen1*.

6. Then, save the *PopupScreen1* image.

7. Next, open the **Screen Event Handling** tab once again (see Figure 2.125). Use the **Add>>** button in Figure 2.125 to add *PopupScreen1.bmp* to the **Image Names** list. Then, select the added image, and click the **Click on Hotspot** check box in Figure 2.125. Finally, click the **Apply** button, and then the **OK** button to save the changes.

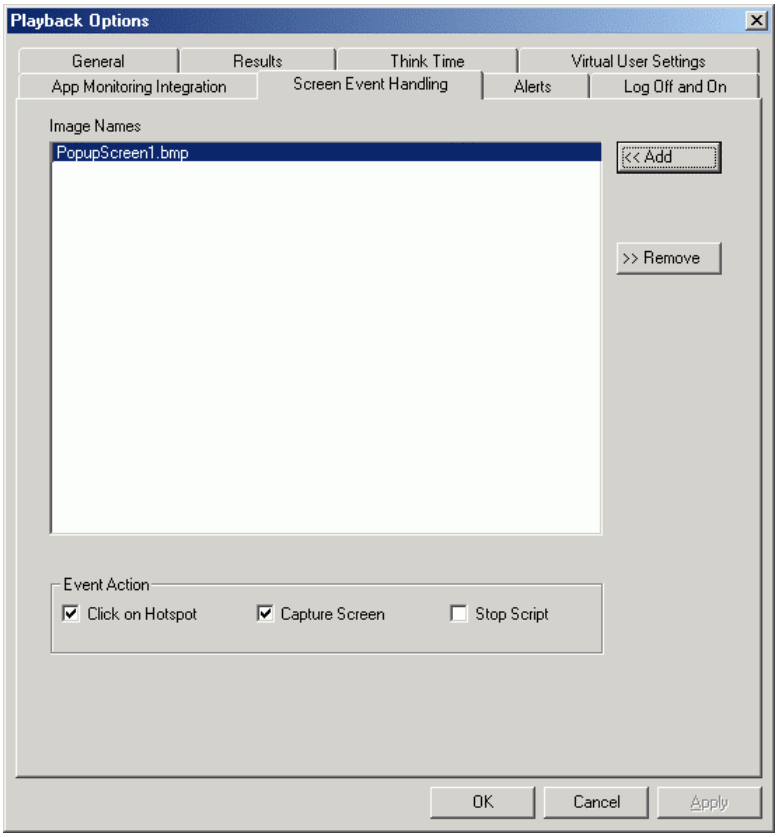

Figure 2.125: Adding a captured image to the Image Names list

Sometimes, you might be aware of the messages that may popup in the course of script execution. For example, if a virus scan is scheduled to take place at 1 PM daily, then you would be aware that the progress tracker will appear at 1 PM every day. This is sure to disrupt script execution. In such a case, you can manually capture the image of the progress tracker, and then proceed in the manner discussed below.

1. Open the **Screen Event Handling** tab (as discussed before) and add the manually captured image to the **Image Names** list (see Figure 2.125).

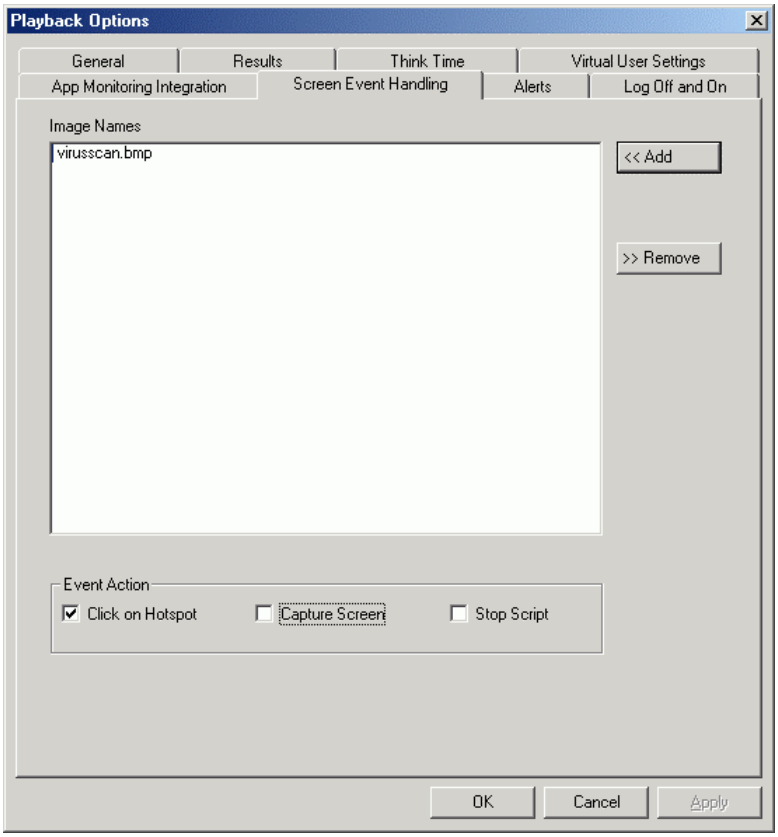

Figure 2.126: Adding the progress tracker's image to the Image Names list

2. Next, select the **Click on Hotspot** check box in Figure 2.126. Finally, click on the **Apply** and **OK** buttons.

2.4.2 Executing scripts from Other Systems

A script can be copied to and played back from systems other than the one on which it was developed. Prior to copying, ensure the following:

- CitraTest's Runtime environment will have to be installed on the system from which the script is to be played back
- The script, when executed, should be able to connect to the application to be monitored. For example, if the **CitrixEx** script that we built earlier is to be copied and run from another system, then such a system should host the Citrix ICA Client, and a shortcut to it should be available on the system's desktop.
- As mentioned previously, CitraTest employs image-recognition and text-recognition techniques to perform script playback. Therefore, the images that will be used by the script during playback, should, in appearance (shape, size, label, etc.), be the same as the ones that were captured. Only then will the script be able to recognize the images and act as instructed. For example, say that a script has been built to connect to the Internet Explorer by clicking on the **Internet Explorer** shortcut on the Windows Desktop. Assume that the whole of this shortcut was captured as an image during script development. If the label of this shortcut is changed to **IE** in the system on which the script is to be played back, then playback will fail.

1. The first step towards playing back scripts from other systems is to bundle the script's dll and exe files into a distributable package. This can be done using the **Package and Deployment Wizard** on the system on which the script was developed. To bundle the required files for the **CitrixEx** project in our example, do the following:

    a. Start the **Package & Deployment Wizard** using the menu sequence depicted by Figure 2.127.

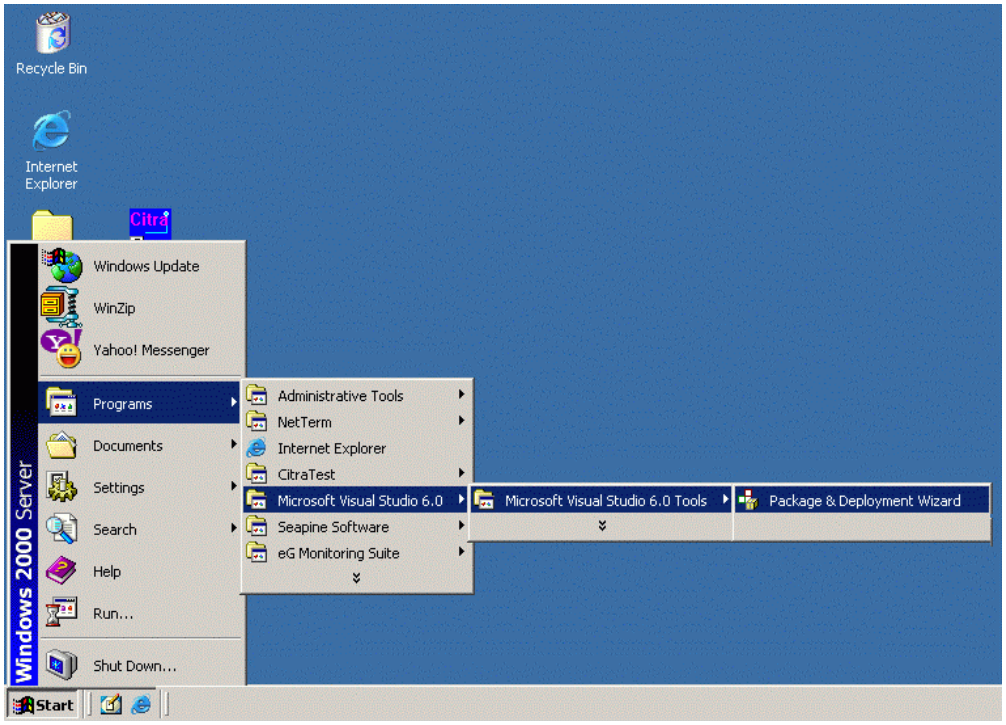

Figure 2.127: Starting the Package and Deployment Wizard

    b. Next, select or specify the complete path to the project (.vbp file) that needs to be packaged. Figure 2.128 reveals that the **CitrixEx** project is being packaged.

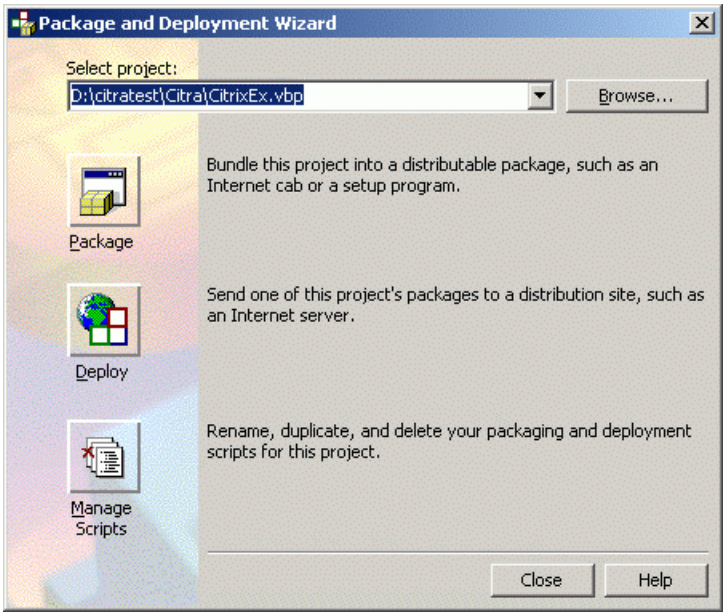

Figure 2.128: Specifying the path to the project to be packaged

c. Next, click on the **Package** button in Figure 2.128 to bundle the **CitrixEx** into a distributable package.

d. The default **Packaging script** is **Standard Setup Package 1** (see Figure 2.129). Leave it as is and click the **Next >** button to proceed with the packaging.

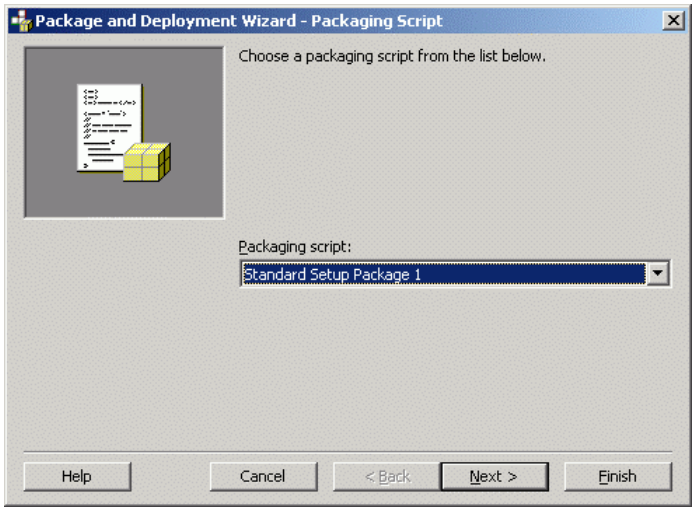

Figure 2.129: Choosing a packaging script

e. Next, select **Standard Setup Package** (see Figure 2.130) as the type of package to be created, and click on the **Next >** button to proceed.

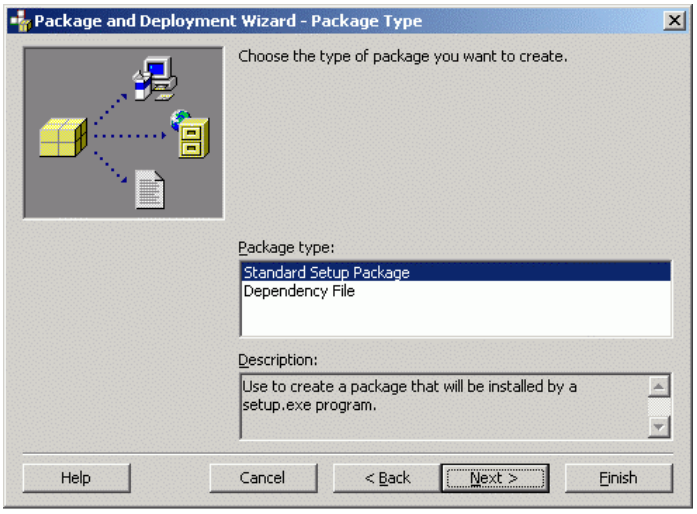

Figure 2.130: Selecting the type of package

f. Specify the path to the folder where the package will be assembled (see Figure 2.131). Note that, for our example, a folder named **Package** has been created for this purpose. The path to the **Package** folder has been provided in Figure 2.131.

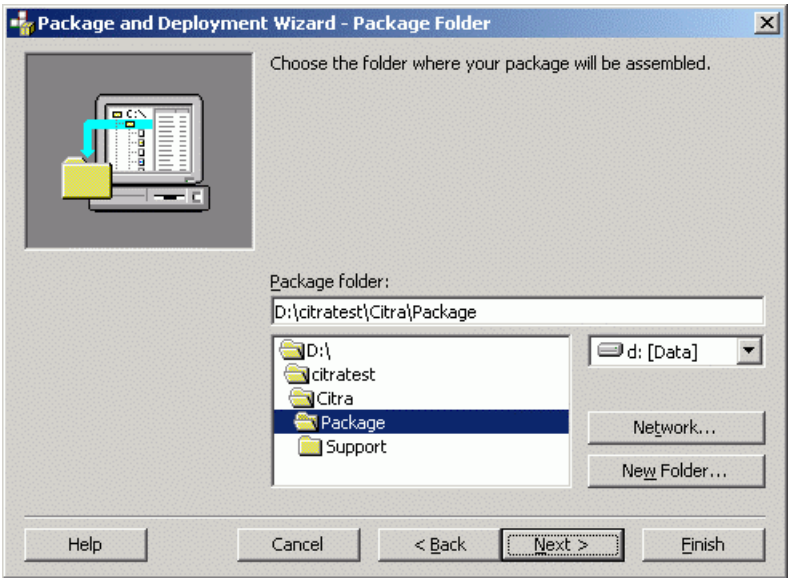

Figure 2.131: Specifying the location where the package will be assembled

g. While the packaging is in progress, you might receive an error message depicted by Figure 2.132. This error will not hamper the packaging process or its output. Therefore, ignore it by clicking on the **OK** button (see Figure 2.132).

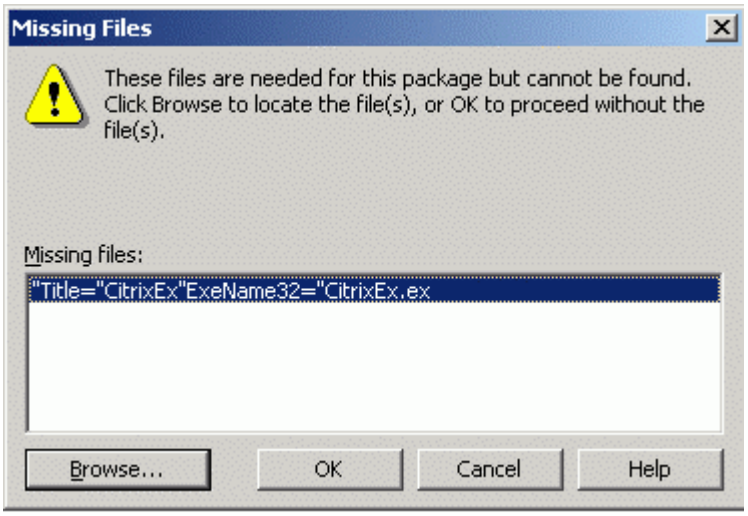

Figure 2.132: Error message

h. Next, Figure 2.133 will appear listing all the files (dll's and exe's) that have been included in the package. To add more files to the list, click on the **Add** button therein. Since our **CitrixEx** package does not include additional files, simply click the **Next >** button to proceed.

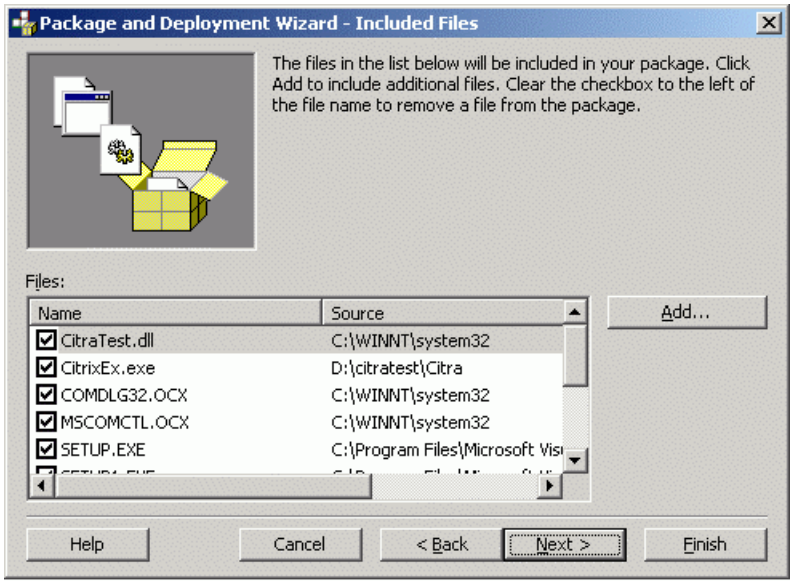

Figure 2.133: Files that form part of the package

i. You can create one large cab file or multiple cab files for a package. For our example however, choose the **Single cab** option, and click the **Next >** button to proceed (see Figure 2.134).

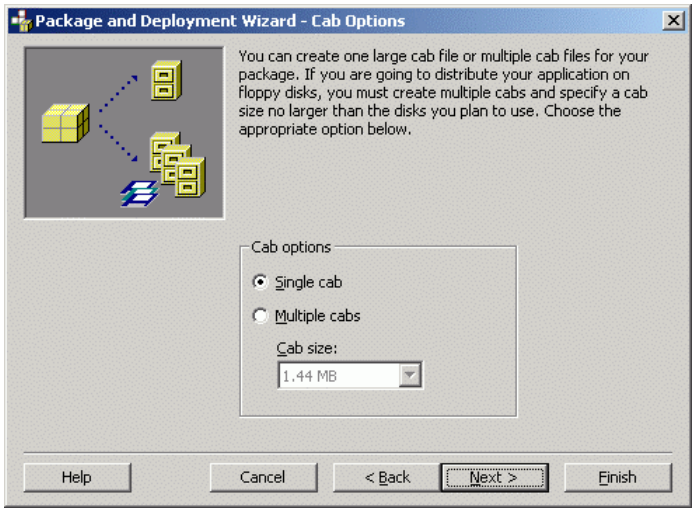

Figure 2.134: Indicating the number of cab files to be created

j. The package will consist of a **Setup.exe**, which when executed on the system on which the script needs to be played back, copies the bundled dll's and exe's to that system. In Figure 2.135 therefore, specify the title to be displayed when the **Setup.exe** is run on the destination system. Then, click the **Next >** button to continue.

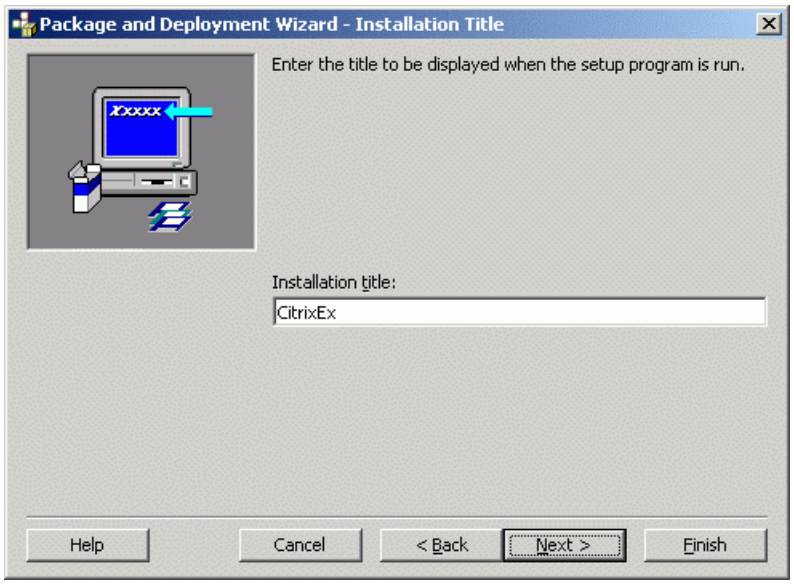

Figure 2.135: Entering the title of the setup program

k. You can change the location to which the bundled dll's and exe's are to be copied by modifying the macros displayed in the **Install Location** column of Figure 2.136. Then, click the **Next >** button to continue.

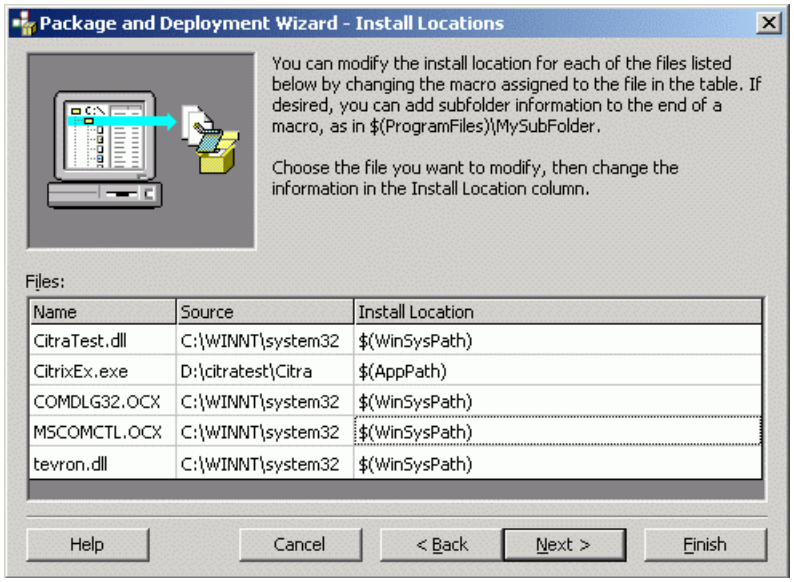

Figure 2.136: Changing the install location of the dll's and exe's

l. Using Figure 2.137 that appears next, indicate whether the listed files are to be installed as shared files or not. If a file is to be shared by another program, indicate the same by selecting the check box against the file name in Figure 2.137. As the displayed **CitrixEx.exe** file need not be installed as a shared file, simply click the **Next >** button to proceed with the setup.

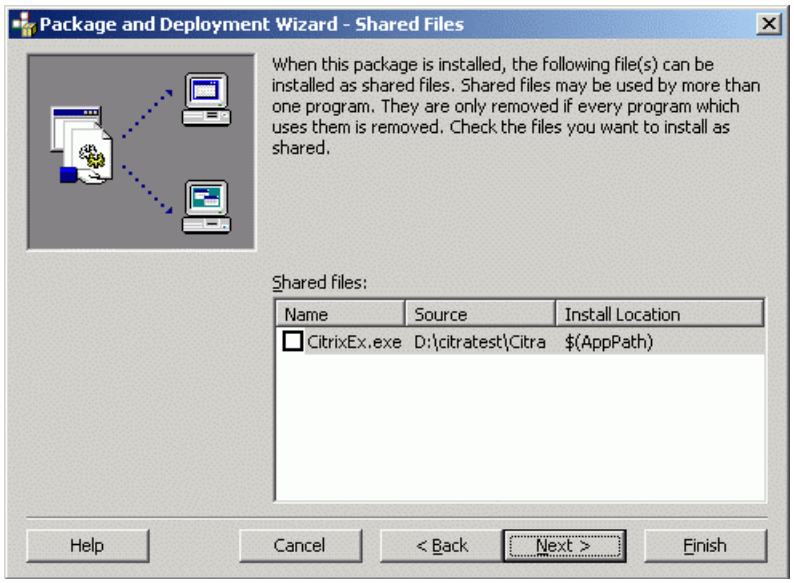

Figure 2.137: Indicating whether files are to be installed as shared files or not

m. Once the wizard is through with the process of collecting the information to build the package, it will request you to provide a name under which the specified configurations need to be saved (see Figure 2.138). Then, click the **Finish** button.

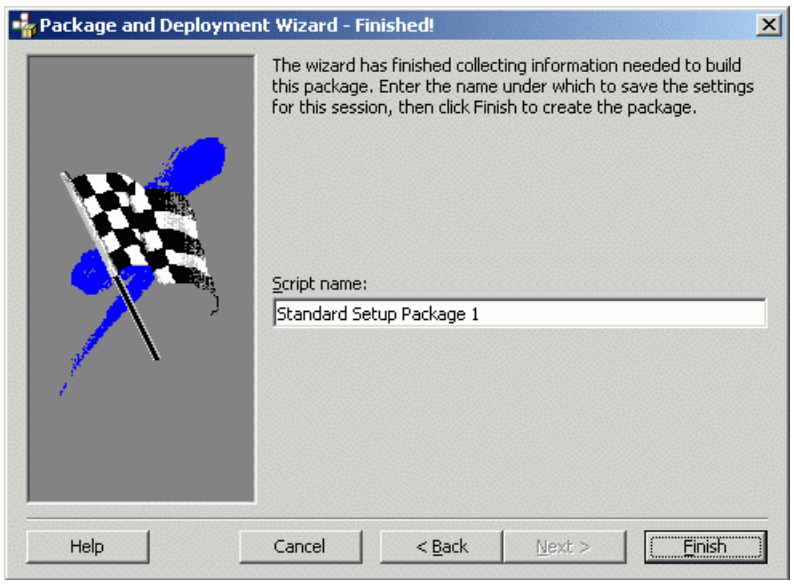

Figure 2.138: Saving the session

n. Upon clicking the **Finish** button, Figure 2.139 will appear, wherein you will have to click the **Close** button to complete the process.

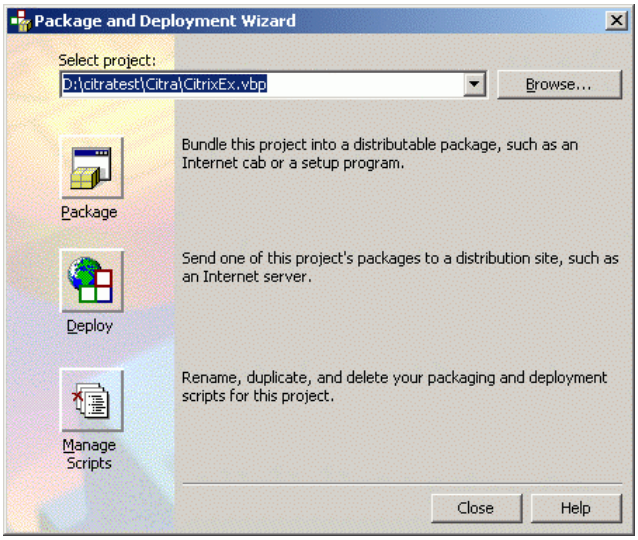

Figure 2.139: Closing the packaging process

o. If you can recall, in our example, a folder called **Package** was created, where the package will be assembled at the end of the packaging process (step 'f'). Figure 2.140 allows you a peek into the **Package** folder, soon after the completion of the packaging process.

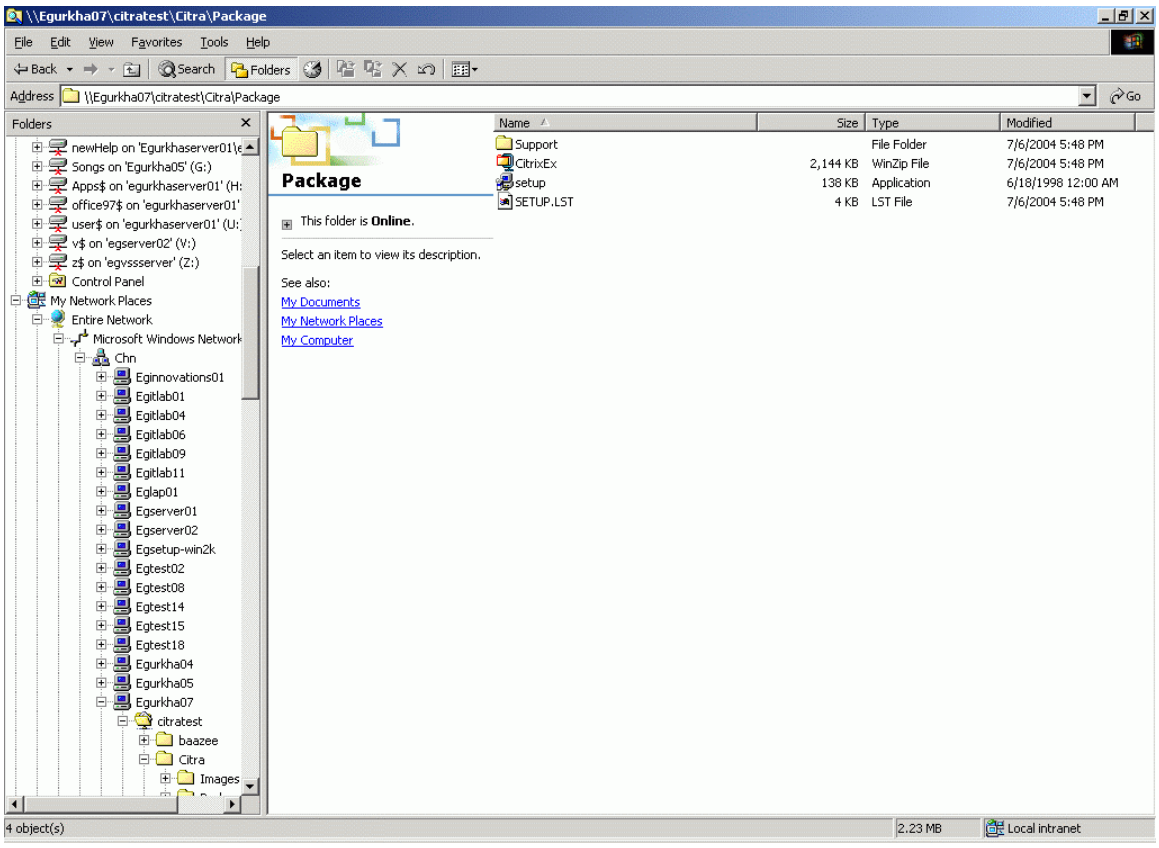

Figure 2.140: The contents of the Package folder

p. The **CitrixEx** winzip file of Figure 158 consists of all the important dll's and exe's that will have to be copied to the destination system. The **setup** application is the executable that will have to be run on the destination system to facilitate the copying.

1. Next, you need to manually copy a few files from the VB project directory in the source system to some location in the destination system. For our example, the following files have to be copied:

   - The Images folder
   - The Package folder
   - The CitrixEx Application
   - CitrixEx.bas
   - CitrixEx.vbp
   - CitrixEx.script

2. Then, proceed to execute the **Setup.exe** on the destination system to copy the bundled exe's and dll's. To achieve this, do the following:

   a. Open the Windows Explorer in the destination system and navigate to the system which hosts the packaged project.

b. From the directory in the source system that contains the packaged project, run **Setup.exe**.

c. The Welcome screen of the setup program will then appear. Note that the title that we had provided earlier (**CitrixEx**) appears here (see Figure 2.141).

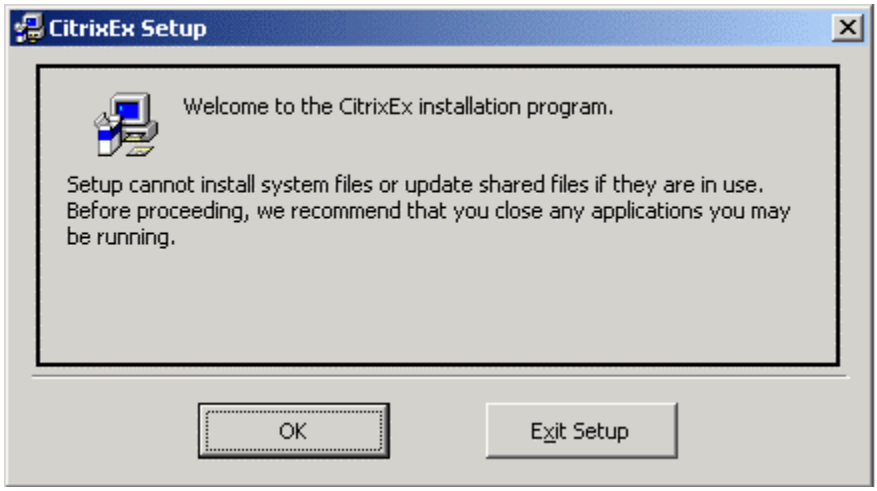

Figure 2.141: Welcome screen of the package installation program

d. Next, provide the directory in which the **CitrixEx** installation has to be saved, and then, click the button at the top of Figure 2.142 to begin installing it.

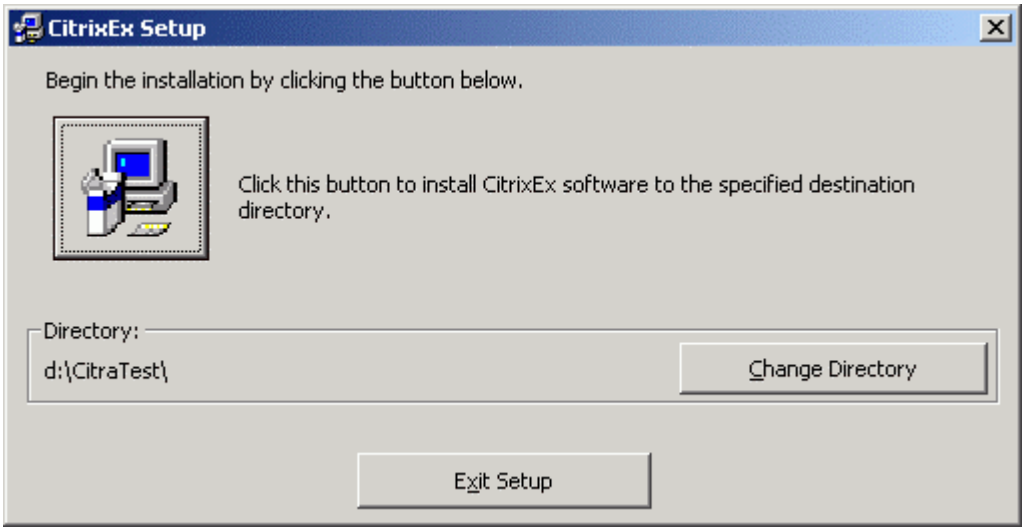

Figure 2.142: Commencing installation of the software

e. Setup will then provide you with the option to add a group for the project to the Programs group of the **Start** menu. Then, click the **Continue** button.

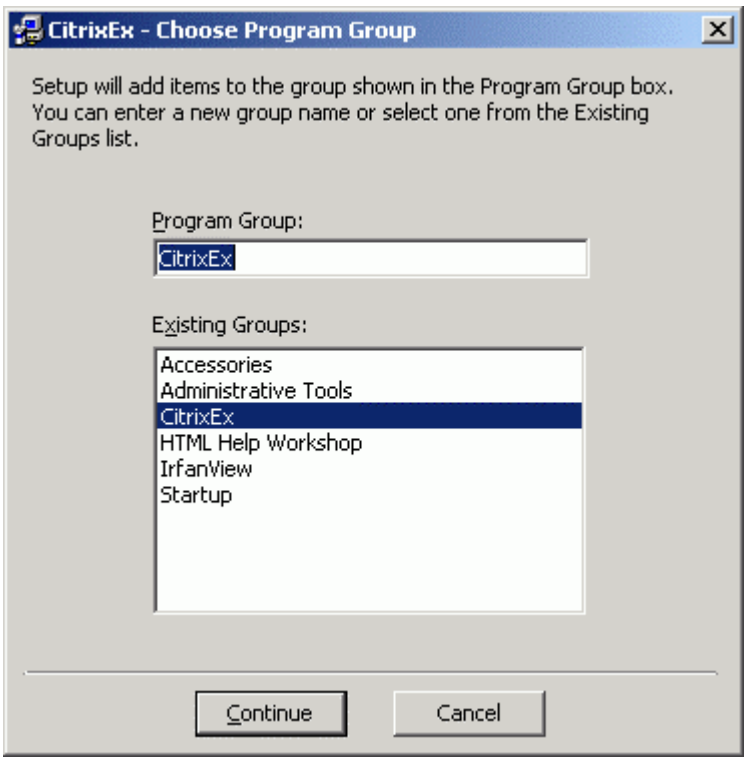

Figure 2.143: Adding a new group to the Programs group

f. Upon completion of setup Figure 2.144 will appear. Click the **OK** button to close the message box.

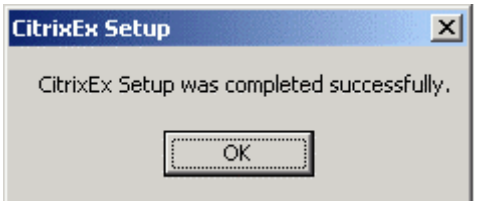

Figure 2.144: Message on completion of setup

g. With that, setup is complete. You can later verify whether the packaged dll's and exe's have been installed to the locations specified while packaging.

3. Since the location of a few key directories would have changed while copying them from the source to the destination system, you need to update the **CitrixEx.script** file with the corresponding changes. To achieve this, open the **CitrixEx.script** directory that was manually copied to the destination system, and make the required changes to the **`imagepath`, `searchpath`, and `fontpath`** parameters (see Figure 2.145).

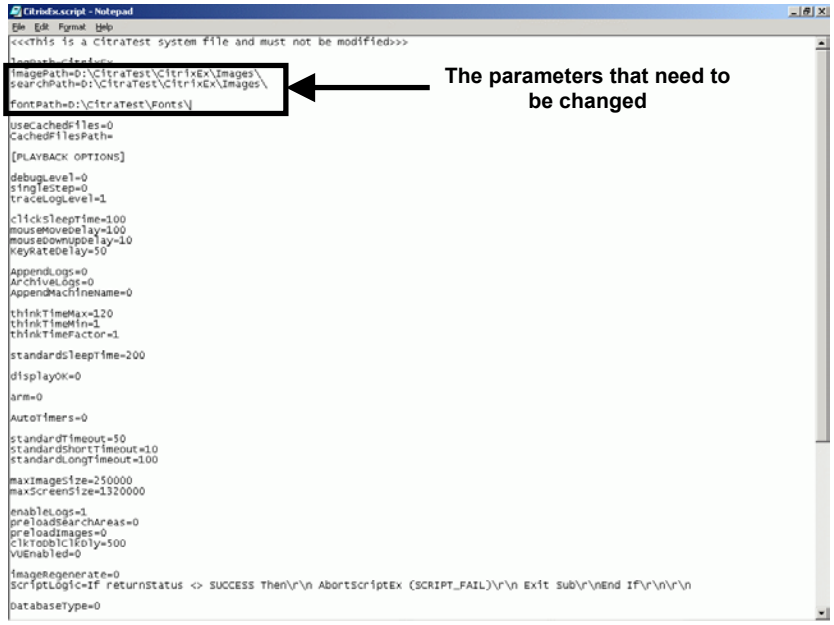

Figure 2.145: Change the image, search, and font paths

The **Fonts** directory will typically exist in the install directory of the CitraTest Runtime environment.

4. Finally, execute the **CitrixEx** application that was manually copied to the destination system to playback the script.

2.5 Troubleshooting

- If a script fails even after successful compilation, then check whether .NET Framework 1.1 was installed on the host using which the script was developed. If so, then check whether the Service Pack 1 for .NET Framework 1.1 was also installed on the host. Without this service pack, the script will not work properly despite getting compiled successfully. Therefore, ensure that Service Pack 1 is also installed.
- When a script involving ICA sessions fails due to an unexpected error in the session, then subsequent executions of the script will also fail. This is because, when a script on ICA sessions fails for the first time, the point of failure of the script is retained. During subsequent executions of the script therefore, the ICA session opens at exactly the point of failure, causing the script to fail yet again.

  To solve this issue, ensure that once a script abnormally terminates, it logs off from the ICA session. Logging off closes all the applications, so that, the next time the script executes, it opens the ICA session at the right place.
- If recording and playback both are attempted on the Terminal Client, a session should remain open for the proper execution of the script.
- To make sure that script execution is not disturbed, it is recommended that the system on which the script executes is left unlocked, and the screensavers disabled. Alternatively, you can configure the script to automatically log off the system when script execution completes, and log back in before the script begins executing. Screensavers can also be controlled in a similar manner. To ensure this, do the following:

a. Open a new script window (see Figure 2.146) and then select **Playback Options** from the **Playback** menu on its menu bar.

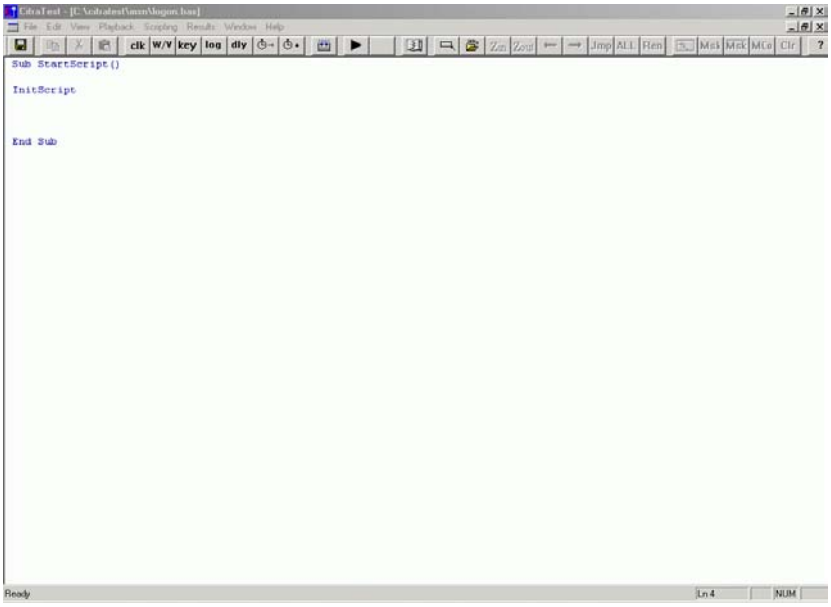

Figure 2.146: Opening a new script window

b. Select the **Log Off and On** tab in the **Playback Options** dialog box (see Figure 2.147). Next, select the **Logon at Script Start** check box and proceed to provide a valid **User Name** and **Password** using which this script can login to the system. Then, specify a name for the **User ID and Password file** that will store the authentication information. A file with the specified name and extension *'.pwd'* will be created, by default, in the same directory as the script.

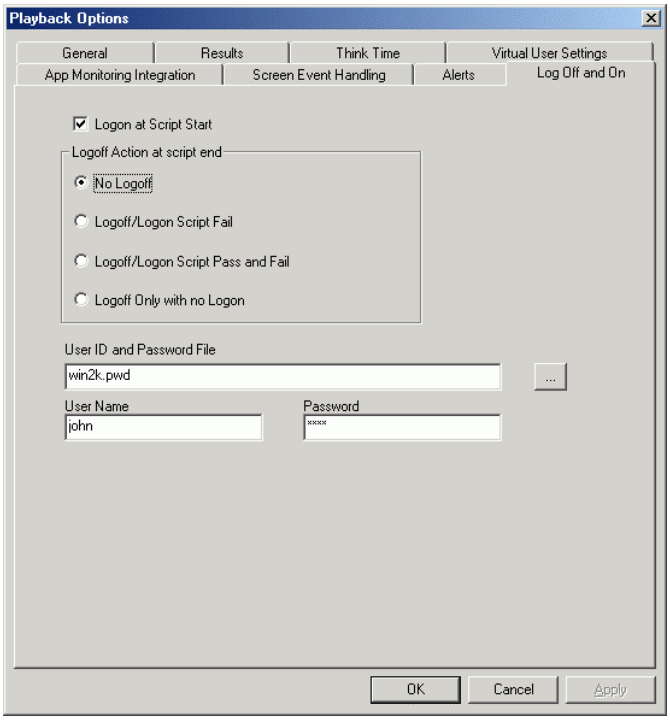

Figure 2.147: Setting the Log Off and On options

c. Finally, click the **Apply** button and then the **OK** button in Figure 2.147 to apply the changes, and then save the newly created script. With that, a Login script has been created.

d. Now, create another new script for disabling screen savers. In the script window that appears, provide the script depicted by Figure 2.148 below and save the changes. This script will now ensure that the screen savers on the system are disabled.

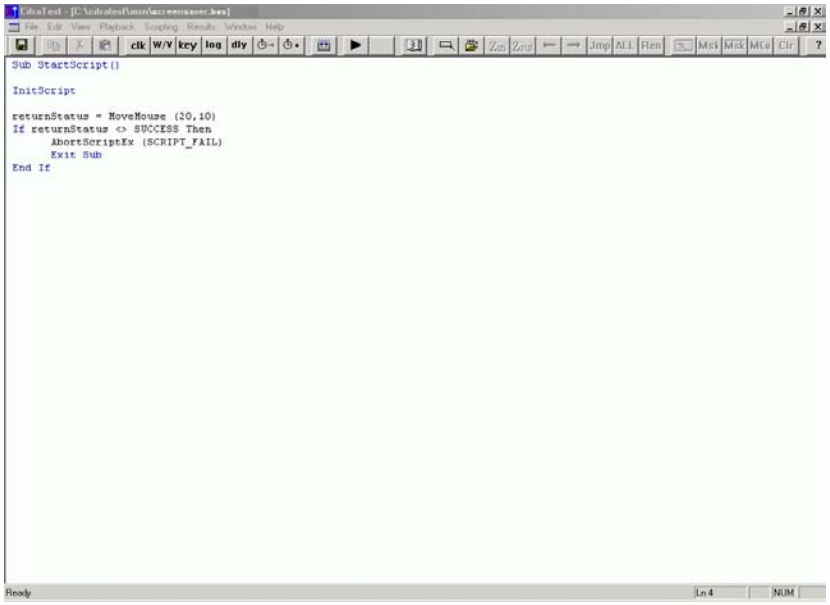

Figure 2.148: Script for disabling screen savers

e. Next, open the main script (i.e., the script that actually emulates a request to a client) and once again, select **Playback Options** from the **Playback** menu on its menu bar. As before, select the **Log Off and On** tab in the **Playback Options** dialog box (see Figure 2.149). Then, choose the **Logoff Only with no Logon** option, click on the **Apply** button therein, and then the **OK** button. This will make sure that once the script execution ends, the system logs off.

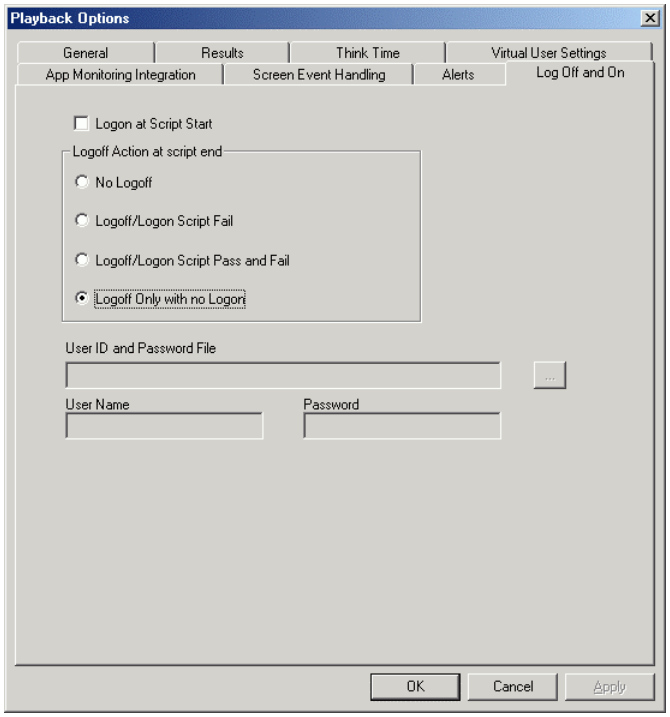

Figure 2.149: Logging off the system

f. Next, at the top of the main script, invoke the script which disables screen savers. Finally, save the main script.

g. Then, while configuring **CitraTest** using the eG manager's admin interface, provide the path to the main script and the login script against the **SCRIPT** parameter, separated by a comma. Remember to mention the path of the main script first, followed by the login script, so that the scripts execute in the same order. Finally, save the configuration.

When the main script begins executing on the target server, it first disables the screen saver on the target server and then proceeds to perform its other functions. Upon completion of execution, the system logs off as programmed. Next, the login script will start running. Using the specified user name and password (see Figure 2.147), this script will log back into the system.

Chapter 3

Conclusion

The eG suite of products has been specially designed keeping in mind the unique requirements of IT infrastructure operators. For more information on the eG family of products, please visit our web site at www.eginnovations.com.

This document has described the purpose, benefits, and procedures involved in the usage of the eG Client Emulator.

For more details regarding the eG architecture and the details of the metrics collected by the eG agents, please refer to the following documents:

- *A Virtual, Private Monitoring Solution for Multi-Domain IT Infrastructures*
- *The eG Installation Guide*
- *The eG User Manual*
- *The eG Measurements Manual*
- *The eG Customization Manual*

We recognize that the success of any product depends on its ability to address real customer needs, and are eager to hear from you regarding requests for enhancements to the products, suggestions for modifications to the product, and feedback regarding what works and what does not. Please provide all your inputs as well as any bug reports via email to support@eginnovations.com.