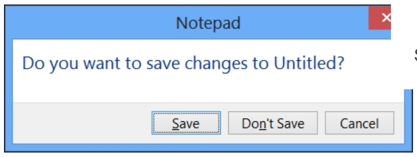
Creating and using a recovery scenario

UFT also enables us to define a specific behavior to handle a wide ar of situations in advance. For example, instead of having to check ou code for the existence of an unhandled pop-up dialog that may interf with the normal flow of our test, we can use a recovery scenario to instruct UFT what to do if such is found. For example, we may tell U to close the dialog and try to rerun the step that failed due to the existence of the dialog. Another option, would be to tell UFT to exit th current Action or test iteration (in such cases, we must ensure that th test or Action begins in the proper context, that is, with the initial set conditions). A simple case of a pop-up dialog is that of Notepad's warning when we try to close the application without having saved the document:


Notepad

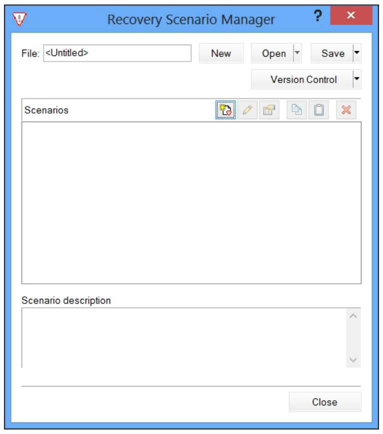
Do you want to save changes to Untitled?

We can use a recovery scenario to handle the pop-up dialog, and this way, we will not have to refer to it in our code. The recovery scenario mechanism is a trap for the events we define as interfering with our test, which we call exceptions.

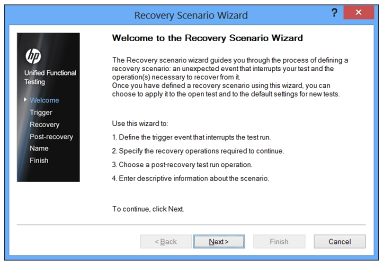
How to do it...

Proceed with the following steps:

1. From the home page of UFT, navigate to **Resources** | **Recovery Scenario Manager**. A dialog with the same title will open. Here, in the scenario frame, click on the **New** scenario button on the left-hand side of the toolbar:



The **Recovery Scenario Wizard** window will then open up:



Welcome to the Recovery Scenario Wizard

The Recovery scenario wizard guides you through the process of defining a recovery scenario: an unexpected event that interrupts your test and the operation(s) necessary to recover from it.
Once you have defined a recovery scenario using this wizard, you can choose to apply it to the open test and to the default settings for new tests.

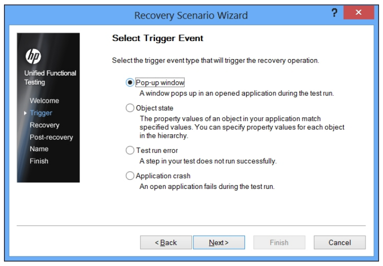
Use this wizard to:

1. Define the trigger event that interrupts the test run.
2. Specify the recovery operations required to continue.
3. Choose a post-recovery test run operation.
4. Enter descriptive information about the scenario.

To continue, click Next.

As it explicitly states, this wizard will guide you through the process of defining a recovery scenario for an unexpected event that needs to be handled, as it impedes your test from proceeding as planned. The wizard goes through four main stages.

2. First, we define the trigger event that interrupts the test run, which can be the appearance of an unexpected pop-up dialog, an object state, an unhandled test run error, or an application crash, as shown in the following screenshot:

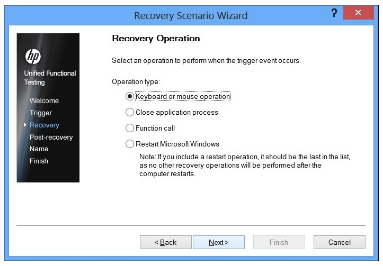


Select Trigger Event

Select the trigger event type that will trigger the recovery operation.

● Pop-up window
A window pops up in an opened application during the test run.
○ Object state
The property values of an object in your application match specified values. You can specify property values for each object in the hierarchy.
○ Test run error
A step in your test does not run successfully.
○ Application crash
An open application fails during the test run.

3. Next, we specify the recovery operations required to continue, which can be a keyboard or a mouse operation, by closing an application process, calling a custom function, or even restarting windows:



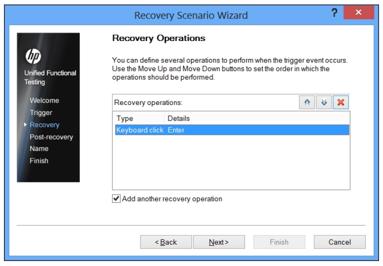
Recovery Operation

Select an operation to perform when the trigger event occurs.

Operation type:

● Keyboard or mouse operation
○ Close application process
○ Function call
○ Restart Microsoft Windows
Note: If you include a restart operation, it should be the last in the list, as no other recovery operations will be performed after the computer restarts.

4. For the sake of this example, we selected **Keyboard or mouse operation**, and pressed the *Enter* key. The operation is added to the list, as shown in the following screenshot:



Recovery Operations

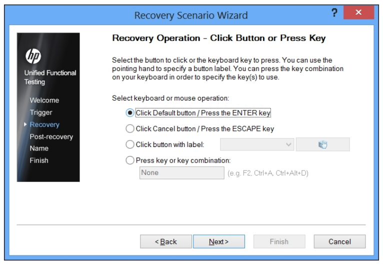
You can define several operations to perform when the trigger event occurs. Use the Move Up and Move Down buttons to set the order in which the operations should be performed.

Recovery operations:

Type	Details
Keyboard click	Enter

✓ Add another recovery operation

Please note that leaving the **Add another recovery operation** checkbox marked allows for repetition of this step. If you need only a single recovery operation, then uncheck the checkbox:



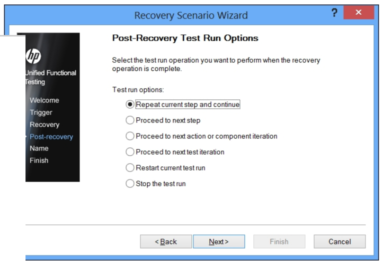
Recovery Operation - Click Button or Press Key

Select the button to click or the keyboard key to press. You can use the pointing hand to specify a button label. You can press the key combination on your keyboard in order to specify the key(s) to use.

Select keyboard or mouse operation:

- [✓ Click Default button / Press the ENTER key]
- Click Cancel button / Press the ESCAPE key
- Click button with label:
- Press key or key combination: None (e.g. F2, Ctrl+A, Ctrl+Alt+D)

5. Next, is a post-recovery test run operation, which is related to UFT's built-in runtime mechanism and tells the tool what to do after the recovery operations are executed. It is imperative to analyze the requirement thoroughly for each particular case, because the options UFT offers would easily fail unless proper care is taken to ensure that test preconditions are kept. For example, there is no point in proceeding to the next test or Action iteration if the required application context is not guaranteed to show up at the start of the iteration. The following screenshot shows various post-recovery test run options:



Post-Recovery Test Run Options

Select the test run operation you want to perform when the recovery operation is complete.

Test run options:

- [✓ Repeat current step and continue]
- Proceed to next step
- Proceed to next action or component iteration
- Proceed to next test iteration
- Restart current test run
- Stop the test run

The **Post-Recovery Test Run Options** are:

- **Repeat current step and continue**: Actually, it is a retry mechanism, which raises the question of what would happen in the case of a recurring error. As no exit condition is provided, an infinite loop of error-recovery-retry-error is a real risk. The same procedure can be rewritten as a loop that repeats the error-causing statement (or block of code) until the error is eliminated. The code is as follows:

```
On Error Resume Next
'Statement that raises error
Do While Err.Number <> 0
'Statement that handles error
'Statement that raises error
Loop
```

- **Proceed to next step**: This test will attempt to execute after the step that gave rise to the error. The code equivalent of this option is the error-handling structure:

```
On Error Resume Next
'Statement that raises error
If Err.Number <> 0 Then
'Statement that handles error
End If
'The next statement
```

Note

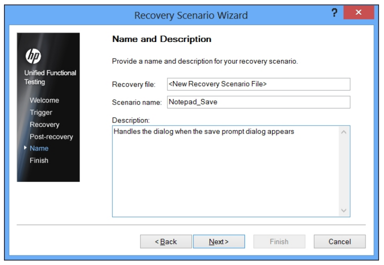
Our test will still fail if the recovery operation does not succeed, or if it does, but for some reason the exact application context fit for the next step is not reached (for example, we handled a pop-up dialog by closing it, but the application opened another one or reopened the same pop-up as it reflected a real problem in the application, such as a script exception as is common on the Web).

- **Proceed to next action or component iteration**: This option allows you to run the next action or component iteration. As previously mentioned, we must ensure that the problem is specific to the case we have run, and that our recovery procedure, together with the iteration initialization, produces the initial conditions required for the next iteration to run.
- **Proceed to next test iteration**: This option allows you to run the next iteration and ensures that the recovery scenario

(for example, closing the application or terminating its process) and the test iteration initialization always produce the conditions for the test to begin running from the first step (for example, opening the application, logging in, and so on).

- **Restart current test run**: This option allows you to restart the current test run. Some isolated glitches might cause a nonconsequential exception, which can be resolved by rerunning the test. However, if it is due to a real bug or some substantial infrastructure problem (for example, the server going down), then this will be to no avail.
- **Stop the test run**: This option allows you to stop the test run, but it enables analysis of the problem without having to review the effects of the exception on the next steps, hence allowing us to focus.

6. Next, we enter descriptive information about the scenario. Name the scenario and save it for future use as shown:



Name and Description

Provide a name and description for your recovery scenario.

Recovery file: <New Recovery Scenario File>

Scenario name: Notepad_Save

Description: Handles the dialog when the save prompt dialog appears

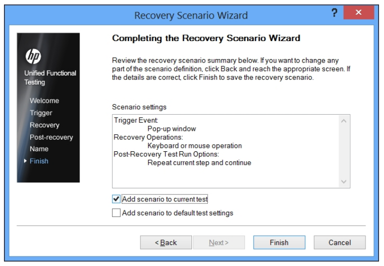
So far, we have used the step names exactly as they appear on screen:

Define the trigger event that interrupts the test run

Specify the recovery operations required to continue

Choose a post-recovery test run operation

7. Next, clicking on the **Next** button will lead to the following finish screen, in which we will check the option **Add scenario to current test**:



Completing the Recovery Scenario Wizard

Review the recovery scenario summary below. If you want to change any part of the scenario definition, click Back and reach the appropriate screen. If the details are correct, click Finish to save the recovery scenario.

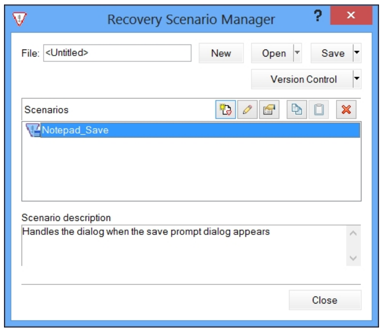
Scenario settings

```
Trigger Event:
            Pop-up window
Recovery Operations:
            Keyboard or mouse operation
Post-Recovery Test Run Options:
            Repeat current step and continue
```

[✓ Add scenario to current test]

Add scenario to default test settings

8. Now, click on **Finish**, which will return to the **Recovery Scenario Manager** dialog box:



File: <Untitled>

Scenarios

Notepad_Save

Scenario description

Handles the dialog when the save prompt dialog appears

9. We will then save the scenario to a file by clicking on **Save**. The resulting QRS file will then be available to all relevant tests.

How it works...

The recovery scenario that we define will be loaded in memory and activated, unless we choose to deactivate it by unloading the attached

ORS file or by using the following code:

```
Recovery.Enabled = False
```

We can set a recovery scenario to be activated on every step or on the occurrence of an error. A warning must be given here, as the first option may carry high costs regarding the performance of the test.

The test will run normally, while in parallel, the recovery mechanism will monitor to check whether the trigger event or events defined in our recovery scenarios actually occur. If they do, then the associated recovery procedure will be called into Action, and after its execution, the post-recovery operation, such as skipping to the next step, restarting the test run, or any of the other options detailed in the previous section, will be carried out.

Recommended / Queue
Feedback (http://community
PREV
Catching errors i…
NEXT
Using a global di…