QTP/UFT Eyes SDK user guide

Installing the QTP/UFT Eyes SDK

1. Download the latest UFT Eyes SDK by clicking on one of the following links:
   a. Users with an on-premise server
   b. Other users
2. Associate the Eyes.fql function library, located in the extracted folder, with your test by navigating to *File* > *Settings* > *Resources* > *+*.
3. If you want Eyes to be included in all tests, make sure to check the *Set As Default* option.
4. Install the `EYES_PATH` environment variable required by the SDK:
   a. After you create the UFT project, go to *File* > *Settings* and click on the *Environment* tab. In the *Variable type* selection box, choose *user-defined*.
   b. Add the variable `EYES_PATH` and assign the full file path of the folder to which you extracted the UFT Eyes SDK.

A sample test

We will now describe a test based on Applitools Eyes step by step. At the end of the article, you will find a full code example.

Test setup

The SDK automatically creates an object instance of the Eyes class called `eyes`. The first step is to assign the eyes ApiKey property the value of your API Key. It is recommended that you store the value in an environment variable called, for example, `MY_APPLITOOLS_API_KEY` and assign it to the property as follows:

```
eyes.ApiKey = Environment.Value("MY_APPLITOOLS_API_KEY")
```

If you have a dedicated server or a private cloud, you need to provide its URL to Eyes. The example below shows the URL hard-coded in the code , but you can choose to define and read an environment variable as described for `MY_APPLITOOLS_API_KEY`:

```
eyes.ServerUrl = "https://myOrganizationeyesapi.applitools.com"
```

The checkpoints of your test will be checked against a reference *baseline*.The baseline name is composed of a test name, an application name, and a runtime environment. We will specify the runtime environment in this step, and the test and application name in the next step. The runtime environment is composed of the names of the operating system, the browser type (or other host application), and the viewport size. You define the environment using one of four variations of the method `SetBaselineInfo`:

```
eyes.SetBaselineInfoFromWindow(Window("WindowsObj"))                ' If the AUT is a window
eyes.SetBaselineInfoFromBrowser(Browser("BrowserObj"),width,height) ' If the AUT is a browser
eyes.SetBaselineInfoFromDevice(Device("DeviceObj"))                 ' If the AUT/DUT is a PerfectoMobile device
eyes.SetBaselineInfo(CustomGuiWindow("CustomAppObj"), "OS", "HostApp") ' In all other cases
```

are:

bj

DOCS

bj

e of the browser object in the Objects Repository.

`width`
The value to be set as the width of the viewport.

`height`
The value to be set as the height of the viewport.

`DeviceObj`
The name of a PerfectoMobile device object in the Objects Repository.

`CustomGuiWindow`
Call to get the custom app object from the Objects Repository.

`OS`
The name of the operating system.

`HostApp`
The host application (e.g. browser): this can be any descriptive string.

Starting the test

You can start the test by calling the method `eyes.Open`, passing as a parameter the application name and test name.These two values, along with the environment information defined in the previous step, uniquely identify the baseline. The values can be any string.

```
eyes.Open "appname", "testName"
```

Typically, a number of tests are grouped together under a single application name. In the Test manager these values are displayed, and you can filter and organize the test results and other information based on these keys.

Making checks

The heart of an Applitools Eyes visual test is executing checkpoints (i.e. capturing a screen shot and sending it to the Eyes Server for matching against the baseline images). The SDK exposes two methods for executing a checkpoint: `eyes.CheckObject` and `eyes.CheckObjectWithTimeout`.

`eyes.CheckObject` is called as follows:

```
eyes.CheckObject object, tag
```

The parameters are:

`object`
An object whose screen image will be captured. You can pass an entire window (e.g. `Window("Notepad")`) or an inner object of the window (e.g. `Window("Notepad").WinEditor("Notepad")`).

`tag`
A descriptive string that describes the checkpoint and is displayed in the Test manager.

There are cases where an image takes time to fully display, and Eyes reports a mismatch for the checkpoint because the screenshot was taken before the image was fully displayed. To help overcome this, Eyes provides the `eyes.CheckObjectWithTimeout` method, which is similar to the `eyes.CheckObject` method but takes an additional `timeout` parameter in milliseconds.

```
eyes.CheckObjectWithTimeout object, "tag", timeout
```

If a checkpoint results in a mismatch, then Eyes will attempt to recapture the checkpoint multiple times until either the checkpoint succeeds or until the `timeout` time has passed.

DOCS

:he results in the Test manager. You can also obtain the results of the test programmatically by checking the value of the global variable `eyesReport` as follows:

```
If Not eyesReport.IsPassed Then
   If eyesReport.IsNew Then
      Reporter.ReportEvent micFail, eyesReport.TestName, "New test inserted, See " & eyesReport.Url & " for details."
   Else
      Reporter.ReportEvent micFail, eyesReport.TestName, "See " & eyesReport.Url & " for details."
   End If
 End If
```

The following properties provide details on the results:

`eyesReport.IsPassed`
: A value of `True` means that all the checkpoints in the test passed. A value of `False` means that at least one checkpoint failed.

`eyesReport.IsNew`
: A value of `True` means that a baseline was not found for this test and the checkpoints of this test were used to define a new baseline. A value of `False` means that an existing baseline was used.

`eyesReport.Url`
: This string contains the URL of the test results. Navigating a browser to this URL will open the Test manager loaded with the results of this test.

So, for example, the batch name may be "nightly build" and this is what is displayed for every run of the batch. But every run a new runId is defined to be the current date and time, and this ID is used in all of the test runs that night, so that the test results are grouped together in the Test manager for that particular batch run.

Advanced commands

In addition to the basic commands described above, the SDK also supports the following features:

- Creating a batch of tests that are displayed together in the Test manager.
- Setting the match level.
- Using branches to mirror software version control systems such as Git.
- Run time log information.

Creating a batch of tests

A *batch* is a collection of tests that are displayed together in the Test manager. The result status of the batch reflects the overall status of the tests in the batch. If all the tests pass, then the batch has a *Passed*status. If any test has a mismatch, then the batch has a *Failed* status.

You can specify the batch a test belongs to using the method:

```
eyes.SetBatch runId, batchName
```

The parameters are:

`batchName`
: A string that is displayed in the Test manager in a list of batch runs.

`tag`
: A unique string that identifies a particular instance of the batch run.

So, for example, the batch name may be "nightly build" and this is displayed for every
atch. But for every run, a new runId is defined to be the current date and
iis ID is used in all of the test runs that night so that the test results are
gether in the Test manager for that particular batch run.

DOCS

to define the *match level* to be used for a test run, for example:

```
eyes.MatchLevel = MatchLayout
```

The following match levels are defined:

- `MatchStrict`: The strictest level - finds any mismatches that are noticeable to the human eye.
- `MatchContent`: Similar to Strict but ignores color differences.
- `MatchLayout`: Checks the relative positions of elements - ignores differences in text or graphics.

For more information on match levels, see How to use Eyes match levels.

Using branches

Many Applitools Eyes users utilize software version control systems such as Git to allow them to maintain multiple versions of their application and to allow concurrent and independent development of new features or bug fixes. Eyes provides a *branching* capability that allows you to maintain multiple versions of a baseline to correspond to multiple version of your code. The version of a baseline in an Eyes branch represents the expected checkpoint images of a particular code branch.

You define which branch a test should run on as follows:

```
eyes.BranchName = branchName
eyes.ParentBranchName = parentBranchName;
```

The `branchName` is a string that identifies the branch. Assigning to `eyes.ParentBranchName` specifes that the value of the string `parentBranchName` is the name of the branch from which `branchName` forks. Assigning to these properties is optional.

When you run a test, Eyes will search for the baseline in the `branchName` branch, if defined. If it isn't defined, then Eyes will use the default branch. If the baseline is not found in `branchName`, then Eyes will search for it in the `parentBranchName` branch. If Eyes doesn't find the baseline there, then it will search for the baseline in the `default` branch. If Eyes still does not find the baseline, then it creates a new baseline, and it is stored in `branchName`.

If you accept changes, add steps to a test, or add new tests and save the baseline, then these are stored in a new version of the baseline on the branch. When you complete development of the branch, you can merge the branch into the parent branch. Baselines that have been added or changed will be copied to the parent branch. For detailed information on how to merge a branch into its parent branch, see The Compare and merge branches page.

Saving runtime log information

The SDK can write logging information to a file. This log can be helpful to Applitools when troubleshooting issues. To save the log to a file, assign the `Eyes.LogFile` property as shown below. The folder structure must already exist.

```
Eyes.LogFile = "c:/logs/applitoolseyes/eyes.log"
```

Example code

```
o include 'Eyes.qfl' via File -> Settings... -> Resources

Applitools Eyes api key
"YOUR API KEY"
```

DOCS

```
- For example if you have a 'Notepad' object in your object repository
Window("Notepad")
eyes.SetBaselineInfoFromWindow(notepad)

' Change the default match level if you want
eyes.MatchLevel = MatchContent

' Start visual testing
eyes.Open "Notepad", "QTP Notepad Test"

' First checkpoint
eyes.CheckObject notepad, "Checkpoint 1"

' Make a change in the application and then validate it using the second checkpoint
notepad.WinEditor("Edit").Type "Testing notepad with Applitools Eyes!"

' Second checkpoint
eyes.CheckObject notepad, "Checkpoint 2"

' All checkpoint executed close the test and get the results into the global variable eyesReport
eyes.Close()

' Report the results using the predefined variable eyesReport
If Not eyesReport.IsPassed Then
   If eyesReport.IsNew Then
      Reporter.ReportEvent micFail, eyesReport.TestName, "New test inserted, See " & eyesReport.Url & " for details."
   Else
      Reporter.ReportEvent micFail, eyesReport.TestName, "See " & eyesReport.Url & " for details."End If
End If
```