Николай Колесник, Геннадий Алпаев

# Учебник по SilkTest

## Практическое руководство

2006
http://silktutorial.ru/

# Оглавление

# Введение

Данное руководство описывает основные принципы работы с программой автоматизированного тестирования SilkTest.
Предполагается, что читатель знаком с основами объектно-ориентированного программирования (ООП), тестирования и разработки программного обеспечения.
Цель данного пособия: дать толчок начинающему изучать SilkTest, поэтому не надейтесь найти здесь исчерпывающее руководство.
Наиболее полным источником информации является Help, поставляемый с программой (на английском языке). Также предполагается хотя бы минимальное знание английского языка.
Все примеры делались на SilkTest версий 6.5, 7.1 и 7.5.

Все примеры, представленные в книге, вы можете найти в прилагающемся архиве по адресу http://silktutorial.ru/TestApp.zip.

## Начало работы с SilkTest

Прежде чем начинать работу, необходимо изучить базовые понятия SilkTest. Скрипты в SilkTest состоят из двух частей: фрейм и тесткейс.

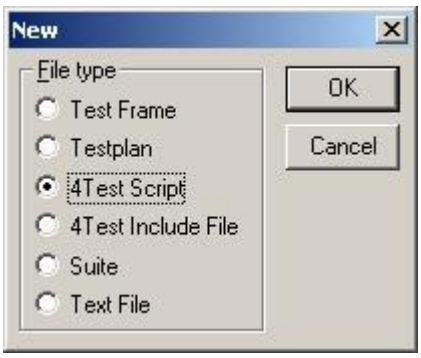
- **Фрейм (Frame)** – это описания окон, с которыми работает SilkTest. Для каждого окна, которое будет использоваться в работе, необходимо создать описание (это делается автоматически). Подробно работа с фреймами описана в главе 1. Использование фреймов (Frame)
- **Тесткейс (Testcase)** – собственно код программы, в котором описываются действия с системой (в частности, например, с окнами)

Информация об окнах обычно хранится в файлах с расширением ***.inc**, тесткейсы – в файлах с расширением ***.t**. В нашем первом примере для простоты мы поместим все в один файл с расширением ***.t**, хотя вообще делать это не рекомендуется (исключение можно сделать лишь для очень маленьких проектов, в которых нет смысла разбивать проект на несколько файлов).

В качестве примера для работы в нашем руководстве мы выбрали приложение TestApp, которое поставляется вместе с SilkTest.

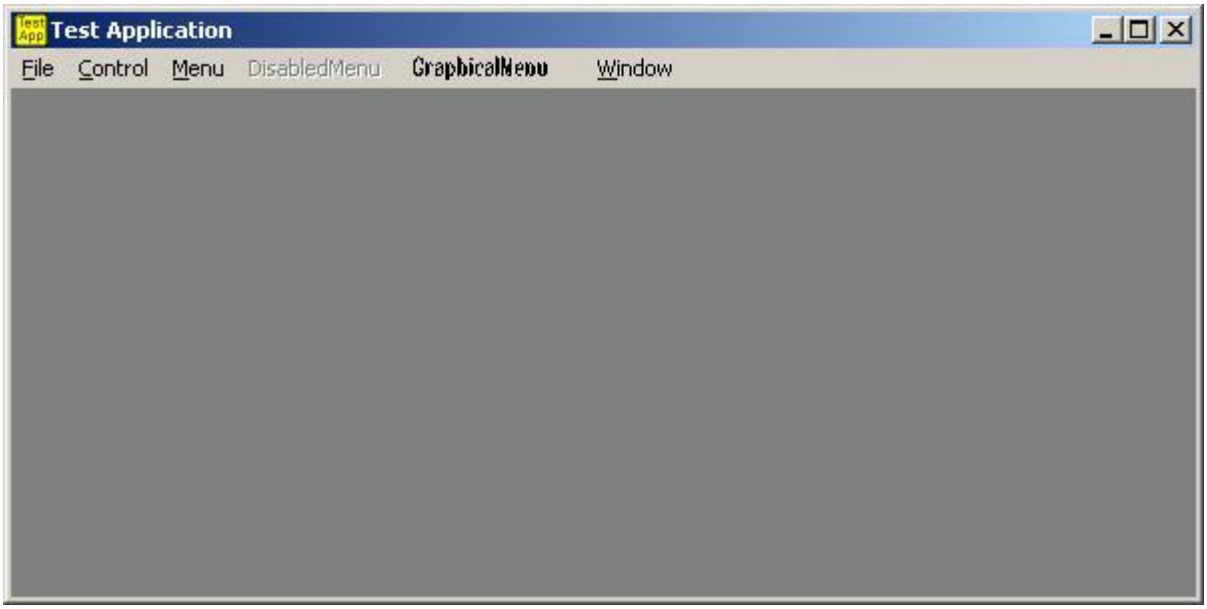
## Создание фрейма приложения

Итак, прежде всего создадим файл, в который затем будем помещать описания окон и скрипты. Для этого в SilkTest выбрите пункт меню *File - New*, в открывшемся окне выберите пункт 4Test script и нажмите OK.

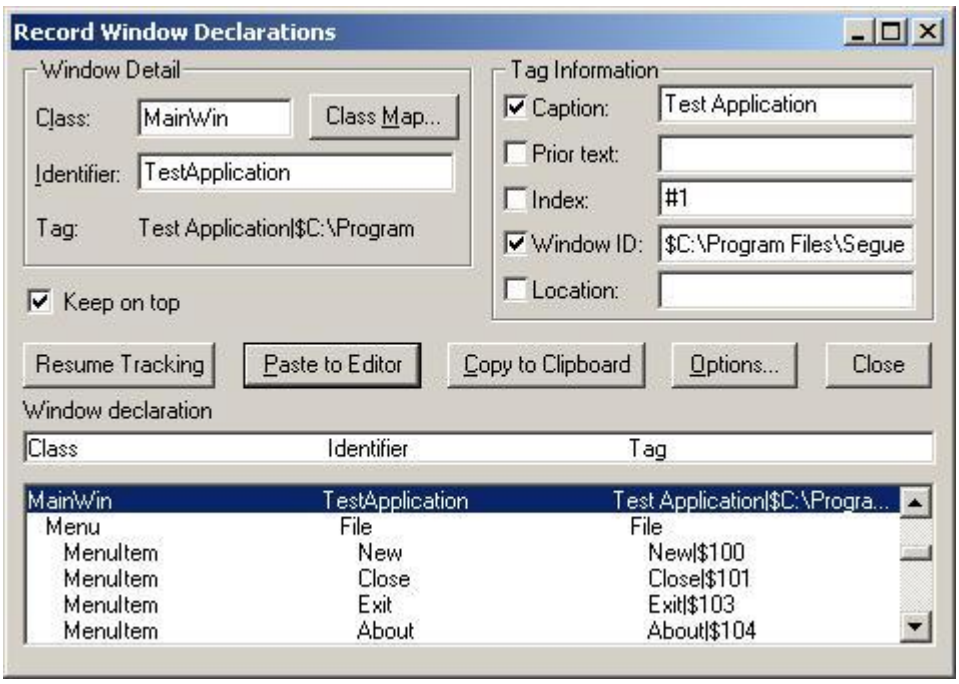



У вас откроется пустое окно, в которое можно помещать описания окон и скрипты. Желательно сразу сохранить файл, так как в случае написания кода вручную в несохраненном файле не будут работать функция Автозаполнения (выпадающий список с перечислением доступных свойств и методов) и всплывающие подсказки для функций (с перечислением необходимых аргументов). Для сохранения файла выберите пункт *File - Save* и сохраните файл.

SilkTest подготовлен к работе. Теперь необходимо подготовить наше тестовое приложение. Его можно открыть из главного меню *Пуск - Программы - SilkTest - Sample Applications - TestApp* Либо запустить файл testapp.exe из папки, в которую установлен SilkTest (*"C:\Program Files\Segue\SilkTest"* для старых версий SilkTest, либо *"C:\Program Files\Borland\SilkTest"* для SilkTest версии 8.0 и выше). Тестовое приложение запущено.




Теперь приступим к записи описаний окон (или, как их еще называют, оконных деклараций). Для этого в SilkTest выберем пункт меню *Record - Window Declaration,* дождемся появления окна Record Window Declaration, затем наведем курсор на заголовок окна TestApp и нажмем комбинацию клавиш *Ctrl+Alt*. В результате окно Record Window Declaration примет такой вид:




В нижней части окна мы видим список всех объектов приложения, расположенных в таком же иерархическом порядке, как они расположены в самом приложении. Выделяя объекты в этом списке в верхней части окна мы видим, как их распознает SilkTest.

В данном случае выделено собственно главное окно приложения. В разделе *Window Detail* мы видим, что это окно является окном класса MainWin (в зависимости от того, к какому классу принадлежит окно или элемент управления, с ним можно выполнять разные действия). Идентификатор окна (в нашем случае это TestApplication) – это, фактически, имя этого окна. Ниже мы видим поле *Tag*. Тег – это строка, уникальная характеристика окна, по которому SilkTest распознает элементы управления в приложениях. В правой части окна Record Window Declaration мы можем посмотреть более подробную информацию о теге. В нашем случае тег состоит из заголовка (Caption) и идентификатора (Window ID). Кроме того, у этого окна есть индекс (Index), в нашем случае он равен "#1", однако, так как соответствующая галочка возле него выключена, он не будет присутствовать в теге.

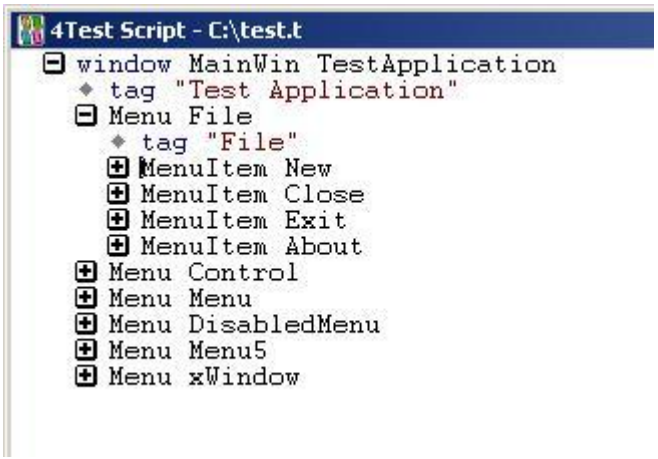
Собственно сейчас уже можно вставлять декларацию окна в документ SilkTest, однако прежде чем сделать это, мы кое-что настроим в системе распознавания окон.

Нажмите кнопку *Options*, отключите опцию *Record Multiple Tags* в нижней части окна *Options*, в разделе *Default tag* выберите **Caption**, в разделе *Window declaration* identifiers выберите **Use the Caption** и нажмите **ОК**.


Теперь в окне Record Window Declaration нажмите кнопку *Resume Tracking* (возобновить запись), снова наведите курсор мыши на приложение TestApp и нажмите *Ctrl+Alt*. Как видно, окно Record Window Declaration слегка изменилось. Теперь в качестве тега может выступать только один идентификатор (например, Caption), а не несколько, как было в прошлый раз.

Более подробно о тегах и о том, почему именно мы рекомендуем отключать опцию Record Multiple Tags можно прочитать в главе 1.2.2 Работа с тэгами (tags).

Теперь можно нажать кнопку *Paste to Editor* и описание окна TestApp вставится в документ SilkTest.




В полученном примере:

- MainWin, Menu и MenuItem – это классы, по которым SilkTest различает, как работать с тем или иным элементом
- TestApplication, File, New, Close (черного цвета) – это имена объектов приложения. Имена объектов можно (а иногда даже нужно) менять. Например, имя TestApplication кажется слишком длинным, его можно заменить на TestApp
- "Test Application", "File" (значения, содержащиеся в кавычках) – это теги объектов. По тегам SilkTest различает элементы управления одного класса. Например, пункты меню File, Control и Menu5 находятся на одном уровне иерархии (что естественно), а значит у каждого из них должен быть свой уникальный тег

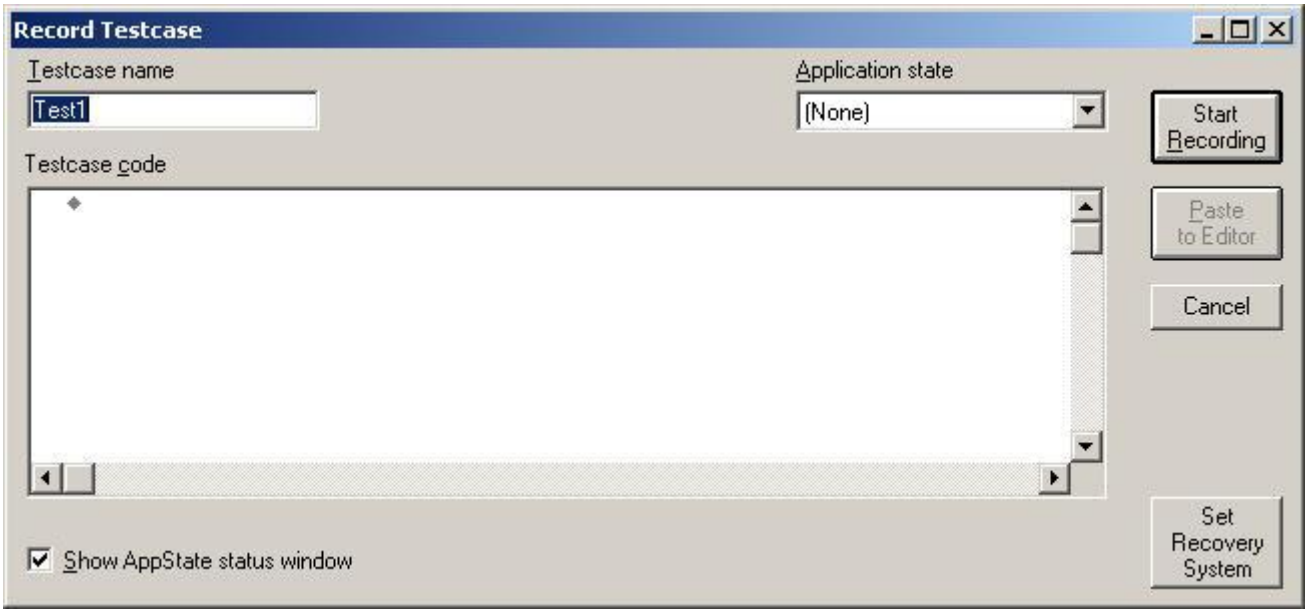
## Запись и воспроизведение скрипта

Когда декларации всех необходимых окон созданы можно приступить к записи и воспроизведению скрипта. Сразу оговоримся, что средства записи в SilkTest лучше использовать только на первых порах, знакомясь с приложением. В дальнейшем вам, скорее всего, придется перейти полностью на ручное написание скриптов. Однако пусть это вас не пугает: при создании сложных скриптов обычно гораздо проще писать все руками, чем сделать запись, а потом вносить изменения.

В качестве примера скрипта мы используем следующий сценарий:

1. Выберем пункт меню *File - New*
2. Передвинем главное окно приложения
3. Выберем пункт меню File - New
4. Закроем главное окно TestApp

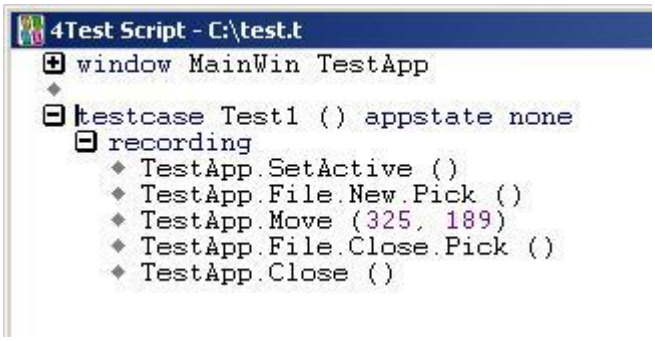
Для начала записи выберите пункт меню *Record - Testcase*. На экране появится окно *Record Testcase*




В поле *Testcase* name введите имя теста (или оставьте существующее), в выпадающем списке *Application state* выберите пункт **(None)** и нажмите кнопку *Start Recording*.

После этого на экране появится небольшое окошко *Record Status*, в котором отображается статус записи и текущее окно, находящееся под курсором мыши. Кроме того, есть кнопка *Pause*, позволяющая приостановить запись, и кнопка *Done*, останавливающая процесс записи тесткейса.

Выполните в приложении TestApp действия, перечисленные выше, и нажмите кнопку *Done* в окне *Record Status*. На экране снова появится окно *Record Testcase*, однако на этот раз с записанным скриптом. Нажмите кнопку *Paste to Editor* и записанный код вставится в документ SilkTest.




Давайте проанализируем полученный код.

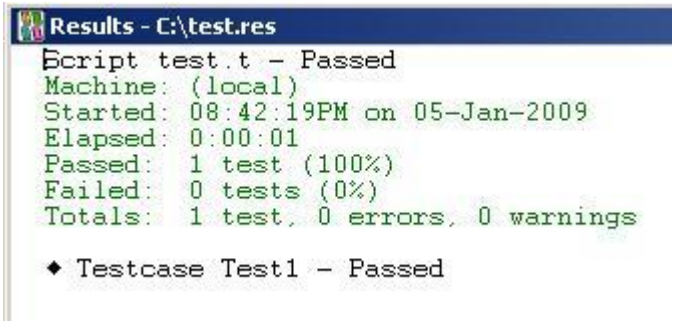
- Скрипт начинается с ключевого слова **testcase**. Далее следует его имя (Test1), в скобках можно передать параметры для этого тесткейса. Затем следует ключевое слово appstate со значением none (подробнее об аппстейтах читайте главу 2.2 Что такое testcase, appstate и как с ними работать, сейчас мы не будем углубляться в эту тему)
- Далее следует ключевое слово **recording**, которое показывает, что код ниже был записан автоматически. Это ключевое слово является лишь информативным и его можно смело удалить
- **TestApp.SetActive ()** – активация окна приложения. Хотя SilkTest автоматически активирует окно при первом обращении к нему (или к любому элементу внутри этого окна), все же желательно использовать метод SetActive(), чтобы быть точно уверенным, что окно активировано
- **TestApp.File.New.Pick ()** – выбор пункта меню *File - New*. Метод Pick() используется для выбора пунктов меню (для других элементов управления обычно используется метод Click() )
- **TestApp.Move (325, 189)** – метод Move() используется для передвижения окон по экрану. В качестве параметров в этот метод передаются новые координаты по горизонтали и вертикали
- **TestApp.Close ()** – метод Close() служит для закрытия окон

Обратите внимание на то, как SilkTest обращается к объектам. Если нам нужно обратиться к пункту меню *New*, который является потомком меню *File*, мы не можем написать *TestApp.New*. В этом случае SilkTest выдаст ошибку. В этом примере совсем немного элементов управления и нет большой вложенности объектов (иерархии). Однако в больших приложениях иерархия может быть огромной (10-20 уровней вложенности). В таких случаях необходимо существенно модифицировать фрейм, группируя элементы управления не так, как это делает SilkTest по умолчанию. Об этом подробнее рассказывается в главе 1. Использование фреймов (Frame).

Теперь осталось запустить записанный тесткейс и посмотреть, отработает ли он так, как был записан. Не забудьте открыть приложение TestApp, которое было закрыто в процессе записи!

Для того, чтобы воспроизвести скрипт, необходимо выбрать пункт меню *Run - Testcase*, в появившемся окне выбрать необходимый тесткейс (в нашем случае это Test1) и нажать кнопку *Run*.


Скрипт отрабатывает примерно за 1 секунду и происходящее практически невозможно понять! В конце SilkTest выдаст отчет о проделанной работе (во сколько времени скрипт начал работать, как долго работал, сколько произошло ошибок и предупреждений и т.п.). Более подробно обо всем этом мы поговорим в следующих главах.

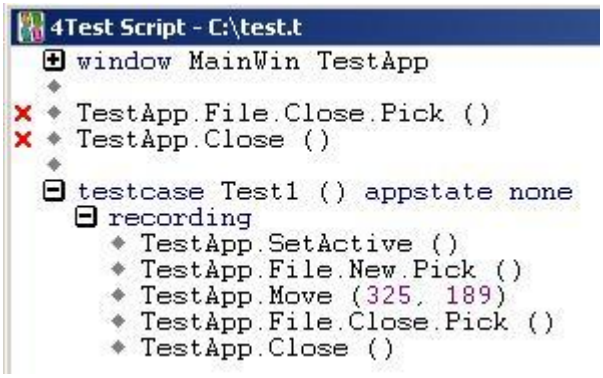



## Несколько слов о редакторе SilkTest

Вы, скорее всего, уже заметили, что редактор в SilkTest отличается от большинства других редакторов кода. И это неспроста. Дело в том, что в отличие от других языков программирования, в языке 4Test используется иерархическое представление блоков кода.

В языке c++, например, блоки выделяются в помощью фигурных скобок **{..}**. В языке Pascal отделение блоков производится с помощью конструкции **begin..end**. В SilkTest для того, чтобы выделить какой-то блок (например, условный блок **if**, или циклы **for**), необходимо поместить код, который должен быть в этом блоке, на один уровень "правее" в редакторе. Точно так же можно скрывать длинные комментарии, оставляя видимой только первую строку и раскрывая весь комментарий по мере надобности.

Если вы поместите какой-то блок кода в недопустимом месте, то SilkTest пометит эти строки красными крестиками, а в строке статуса выдаст сообщение Syntax error.




"Передвигать" блоки текста можно с помощью комбинаций клавиш *Alt + стрелочки* (вверх, вниз, влево или вправо, в зависимости от того, куда вам необходимо передвинуть блок текста). Чтобы "раскрыть" блок текста, нажмите *Ctrl+клавиша плюс* на цифровой клавиатуре. *Ctrl+цифровой минус*, наоборот, свернет блок текста.

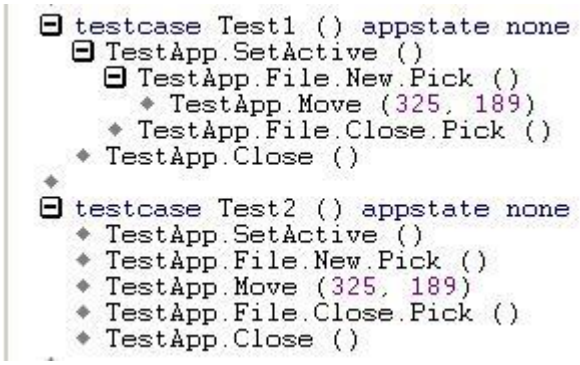
Если вы попытаетесь передвинуть блок слишком далеко влево (например, кусок кода за пределы уровня ключевого слова testcase), то SilkTest просто не даст вам допустить эту ошибку.

В примере с ключевым словом **recording** мы можем избавиться от этого слова таким образом.

1. поставьте курсор на строку, содержащую слово **recording** и нажмите комбинацию клавиш *Shift+стрелка вниз*. Выделится вся строка
2. нажмите клавишу *Delete*. Строка удалится
3. теперь выделите все строки, которые были внутри блока **recording** и нажмите комбинацию клавиш *Alt+стрелка влево*. Весь выделенный текст переместиться на один уровень влево, на котором раньше было ключевое слово **recording**

И хотя последний пункт не является обязательным (равнозначные строки кода могут иметь любой уровень вложенности), все же лучше не делать излишней вложенности там, где в этом нет необходимости.

Например, следующие два тесткейса совершенно одинаковые с точки зрения SilkTest и будут выполняться одинаково хорошо.



# 1. Использование фреймов (Frame)

Существует два способа записи нового окна:

- меню *File - New - Test Frame* (при этом необходимо будет выбрать открытое приложение, для которого создается фрейм)
- меню *Record - Window Declarations...*

В первом случае SilkTest создаст объявление нового окна, а также создаст дополнительное окно MessageBox (это стандартное окно в системе Windows, его вызов присутствует практически во всех программах).

Во втором случае откроется дополнительное окно *Record Window Declarations*, в котором будут отображаться все объекты окна, над которым в данный момент находится курсор мыши. Для того, чтобы остановить процесс сканирования и поместить информацию о текущем окне в документ SilkTest-а необходимо нажать комбинацию клавиш *Ctrl-Alt* и нажать кнопку *Paste to Editor*. После этого в текущий документ SilkTest-а добавятся описания этого окна.

В тех случаях, когда окно небольшое, используется один раз, либо имеет мало объектов можно так и оставить его объявленным как окно (window). Однако в случае больших приложений, либо когда автоматизация тестирования происходит одновременно с разработкой и постоянным расширением программы, имеет смысл объявлять винклассы вместо окон с целью их дальнейшего использования более широко.

## 1.1 Общие сведения о винклассах (winclass)

Винкласс (или просто класс) - это класс окна. В отличие от окна просто обратиться к свойствам и методам винкласса нельзя, необходимо сначала объявить окно этого винкласса.
Общий вид объявления винкласса такой

***[scope] winclass wclass-id [ : derived-class] statements***

где
**- *[scope]*** - необязательное указание того, как будет доступен винкласс: **public** - доступен везде, **private** - доступен только для файла, в котором он объявлен
**- *wclass-id*** - собственно имя винкласса (должно быть уникально в пределах всех модулей, подключенных с помощью ключевого слова use) **- *derived-class*** - класс, от которого наследуется объявляемый винкласс. Наследуются все методы, свойства и объекты родительского класса
**- *statements*** - здесь идут описания собственно объектов класса, его переменных и констант

Пример: Откройте Test Application из поставки SilkTest-а и выберите меню *Control - Check box*. Теперь в SilkTest-е выберите пункт меню *File - New - Test Frame*. Из списка приложений выберите Test Application и на жмите ОК. В файл (по умолчанию Frame.inc, однако его имя и папку, где он будет храниться, можно выбрать в том же окне, где находится список запущенных приложений), как уже говорилось, SilkTest добавил описание двух окон: главного окна (window MainWin TestApplication) и окна сообщений (window MessageBoxClass MessageBox).

Хотя у нас и было открыто окно Check Box в приложении, его описание не попало в файл. Его можно добавить с помощью меню *Record - Window Declaration*. Сделав это, вы увидите, что добавилось описание окна xCheckBox (window DialogBox xCheckBox).

Теперь посмотрим внимательнее на объявления этих окон. Первая часть объявления - это ключевое слово window, которое указывает на то, что начинается описание окна. Далее следует указание, к какому типу (классу) принадлежит это окно (на данный момент мы имеем объявления трех видов: MainWin - главное окно, DialogBox - диалоговое окно, и MessageBoxClass - окно сообщений). Эти винклассы уже описаны в стандартных inc-файлах SilkTest-а, посмотреть их объявления можно в файле winclass.inc в папке *Program Files\Segue\SilkTest*.

Если раскрыть объявление этого окна (нажав на плюсик справа от объявления), то для всех окон мы увидим ключевое слово **tag** (тег) или **multitag** (мультитег). Тег - это характеристика, которая уникально идентифицирует окно, а также любой другой элемент управления. Подробнее о тегах рассказано в главе 1.2.2 Работа с тегами (tags).

У окна xCheckBox, кроме того, имеется строка "parent TestApplication", которая указывает на то, что данное окно имеет предка и указывается имя этого предка. Данная запись позволяет обращаться к этому окну непосредственно, не записывая перед ним имя его предка, например: xCheckBox.SetActive() .

Существует также другой способ: объявить xCheckBox как винкласс, а затем внутри главного окна определить переменную этого класса.

**Code**

```
        [-] TestApp_DialogBox_CheckBox xCheckBox
```

Тогда обращение к этому окну будет выглядеть так: TestApplication.xCheckBox.SetActive() . В этом случае определять предка не нужно (т.е. строку "parent TestApplication" надо удалить.

Какой из способов использовать - решать вам самим. Польза первого способа в том, что он позволяет сократить запись при обращениях к окну, т.к. нет необходимости указывать предка. Однако если одновременно тестируется несколько приложений, то в каждом из них могут оказаться окна с одинаковыми именами и тогда у вас будут проблемы с именованием окон. В этом случае предпочтителен второй вариант.

Пример: опишем новый винкласс для диалогового окна xCheckBox и объявим окно этого типа в двух вариантах.
Первый вариант:

**Code**

```
[-] winclass TestApp_DialogBox_CheckBox : DialogBox
        [ ] tag "Check Box"
        [ ] parent TestApplication
[ ]
[ ] window TestApp_DialogBox_CheckBox xCheckBox
```

Второй вариант:

**Code**

```
[-] winclass TestApp_DialogBox_CheckBox : DialogBox
        [ ] tag "Check Box"
        [ ] parent TestApplication

[-] window MainWin TestApplication
        [+] multitag "Test Application"
                [ ] "$C:\Program Files\Segue\SilkTest\testapp.exe"
        [ ]
        [-] TestApp_DialogBox_CheckBox xCheckBox
```

Небольшое замечание по записи фреймов.
В некоторых случаях при наведении курсора мыши на необходимый объект вы можете увидеть некорректное его описание в окне **Windows Declaration** (например, вместо текстового поля **TextField** будет виден **DialogBox** или **CustomWin**). Это может случиться в том случае, если в окне существует невидимый объект, находящийся на самом верхнем уровне иерархии (так называемый *Z-ordering*, по октором определяется, какой объект будет "выше" или "ниже" относительно других объектов).
Для того, чтобы узнать, как описан нужный вам объект, можно после вставки описания окна выбрать пункт меню *Record - Testcase* и вписать в это поле какой-нибудь текст, после чего нажать кнопку **Paste to Editor** и по появившемуся коду скрипта посмотреть, где именно находится объект и как он называется.

## 1.2 Модификация фрейма

В случае небольших окон, или окон с небольшим количеством элементов управления, возможно и нет нужды в модификации записанного окна/винкласса. Однако в случае больших приложений это совсем не так. Во-первых, имена объектам SilkTest дает, ориентируясь на ближайшие надписи (лейблы - labels), а они не всегда располагаются так, как того хотелось бы. Во-вторых, часто имеет смысл изменить теги объектов, если есть вероятность, что в следующей версии тестируемой программы объекты могут сдвинуться внутри окна, либо когда текст возле объекта может меняться (см. в тестовом приложении окно Static Text, Check Box - в них можно изменить подписи к элементам управления).

### 1.2.1 Именование объектов (controls)

Вот несколько простых правил именования объектов, которые помогут упростить вам жизнь:

1. Давайте объектам осмысленные имена (это же правило применимо и к переменным в программе). Если у вас в окне находится несколько текстовых полей, которые относятся к чему-то одному (скажем, возле первого из них будет стоять надпись *"Даты:"*), а рядом с последним полем будет находиться выпадающий список с подписью *"День недели:"*, то, скорее всего, несколько последних полей будут иметь имена *ДеньНедели1*, *ДеньНедели2* и т.д., а это совсем неудобно. Если вам потребуется заполнить все поля в форме, то вам придется вспоминать, какие из них как называются, а сделать это в случае наличия большого числа описанных окон и/или нескольких приложений трудно. Не ленитесь исправить имена сразу после

записи фрейма: вы сделаете это один раз и больше к этому, скорее всего, не вернетесь;

2. Объектам каждого типа придумайте префиксы и используйте их. Это упрощает доступ к нужным объектам через выпадающий список в редакторе, а также позволяет избежать ненужной путаницы, если в окне имеются несколько объектов разного типа (например кнопка, поле и список), которым логично дать одинаковые имена. Примеры префиксов приведены в таблице 1.1. Кроме того можно упростить этот процесс, написав функцию, которая будет добавлять префиксы автоматически (см. пример такой функции в файле **TestAppUtils.inc** из прилагающегося архива, название функции - **AddPrefixes**).

Таблица 1.1 Примеры префиксов для стандартных видов объектов

| **Префикс** | **Объекты** |
|---|---|
| cb | CheckBox |
| btn | PushButton |
| lst | ComboBox, ListBox, PopupList, ListView |
| rlst | RadioList |
| stat | StaticText |
| scb | ScrollBar |
| edt | TextField |
| plst | PageList |
| sb | StatusBar |
| tb | ToolBar |
| trb | TrackBar |
| ud | UpDown |
| tree | TreeView |
| cw | CustomWin |
| w | MainWin, ChildWin |
| d | DialogBox |

3. Не используйте в качестве имен объектов никакие символы, кроме английских и символа "_". Во-первых, это не рекомендуется делать в принципе, так как некоторые компиляторы не понимают такие имена переменных. Во-вторых, каждый раз, обращаясь к стандартному свойству или методу, вам придется менять раскладку клавиатуры

### 1.2.2 Работа с тегами (tags)

Тег - это характеристика, которая уникально идентифицирует окно или любой другой объект в окне. Тег должен быть уникальным, иначе, если в окне описан объект с тегом, который может относиться к двум объектам, при работе тесткейса сгенерируется исключение такого типа:

***** Error: Window '[MainWin]Test Application' is not unique**
Поэтому правильно выбранный тег позволит реже менять фреймы.

Тег может описываться двумя способами:

- собственно tag
- multitag (мультитег)

Мультитег описывает несколько тегов, которые могут быть у объекта, поэтому мультитег более универсален.

Пример тега:

**Code**

```
[-] window MainWin wTestApp
        [-] tag "Test Application"
```

Пример мультитега:

**Code**

```
[-] window MainWin TestApplication
        [+] multitag "Test Application"
                [ ] "$C:\Program Files\Segue\SilkTest\testapp.exe"
```

Однако конструкция описанного мультитега равносильна следующему описанию:

**Code**

```
[-] window MainWin wTestApp
        [-] tag "Test Application|$C:\Program
Files\Segue\SilkTest\testapp.exe"
```

То есть элементы мультитега можно перечислить в теге, разделив их знаком "|" (вертикальной черты). Поэтому в дальнейшем для простоты речь будет вестись о тегах.

**Совет**

По умолчанию SilkTest настроен таким образом, чтобы записывать мультитеги. Это можно исправить в меню *Options - Recorder - Record Multiple Tags*, или непосредственно во время записи деклараций окна *Record - Window Declarations - Options - Record Multiple Tags*. Там же можно указать, какой именно способ записи тега использовать по умолчанию.

Рекомендуется везде, где достаточной одной характеристики объекта, использовать обычные теги, вместо мультитегов. Кроме того, есть еще один важный довод в пользу обычных тегов. В отличие от мультитегов, в обычных тегах можно использовать

переменные и функции, возвращающие строки. В мультитегах же разрешается использовать только константы. Поэтому если вам необходимо в процессе работы скриптов динамически изменять тег окна, то с мультитегами у вас будет гораздо больше проблем.

Существует 5 способов задания тега объекту:

1. Caption (заголовок или подпись объекта) - это текст, который непосредственно связан с объектом и как его видит пользователь (например, надпись на кнопке, подпись для флажка и т.д.). Это наиболее употребляемый способ описания объектов, поэтому рекомендуется при записи фреймов использовать именно его.
   *Например: tag "OK" - тег для кнопки OK.*
2. Prior text (ближайший текст) - ближайший к объекту текст сверху или слева от него. Задается с помощью символа "^" (крышка) перед текстом.
   *Например: tag "^Ближайший текст:"*
3. Index (порядковый номер) - порядковый номер объекта в окне. Нумерация объектов каждого типа ведется слева направо и сверху вниз. Объявляется с помощью символа "#" (решетка) и номером объекта.
   *Например: tag "#4" - может обозначать четвертую кнопку, или четвертый список, или что-либо еще, в зависимости от того, в объекте какого типа этот тег описан.*
4. Window ID (уникальный номер объекта) - это номер, специфичный для любого GUI-объекта. Задается с помощью символа "$" (знак доллара) и собственно номера объекта.
   *Например: tag "$3156".*
   В процессе разработки и изменения приложения эти номера могут меняться, поэтому их следует использовать в самом крайнем случае, если другие описания неуникальны или же попросту невозможны.
5. Location (месторасположение) - положение объекта относительно его предка, задается координатами X и Y, заключенными в скобки с предыдущим символом @.
   *Например: tag "@(84,12)".*
   Этот способ также рекомендуется применять лишь в самых крайних случаях (например, в случае нестандартного элемента управления), так как положение объекта может меняться еще чаще, чем его Window ID.

Кроме того, в тегах могут использоваться:

1. Символы групповой замены (wildcard) - символ "*" (звездочка) заменяет любое количество символов, символ "?" (вопросительный знак) заменяет один символ. Это нужно в тех случаях, когда часть тега может меняться (например, заголовок окна содержит имя файла).
   Пример тега для окна блокнота: ***tag "* - Notepad"***.
   В этом теге указывается, что первая часть окна может быть любой, но окончание должно быть всегда одинаковым. Другой пример: tag "Button?". В этом теге указывается, что начало заголовка всегда будет одинаковым, а в конце может быть один любой символ (например, "1", ":", "X" и т.д.).

2. Уже упоминавшийся символ "|". В теге перечисляются все варианты тегов, которые может иметь объект.
   Например, для кнопки "Отмена" в окне MessageBox может быть определен такой тег: ***tag "Cancel|Отмена|#3|#2"***.
   Первая часть для англоязычной версии Windows, вторая для русскоязычной, третья

- для окна с тремя кнопками (Да, Нет, Отмена), четвертая - для окна с двумя кнопками (ОК и Отмена). Последние два варианта будут работать для любой версии Windows. Здесь необходимо учесть, что ставить тег #2 перед тегом #3 нельзя, так как в случае окна с тремя кнопками этот тег будет использован для кнопки "Нет".

3. Непосредственное указание класса объекта. Указывается имя класса, заключенное в квадратные скобки ("[" и "]"), ставится перед собственно тегом. Например для той же кнопки тег будет выглядеть так:
   ***tag "[PushButton] Cancel|Отмена|#3|#2".***
   Такие конструкции используются, если в окне есть два объекта с одинаковыми тегами и этот объект непосредственно не описан в окне, а лишь является составной частью тега (см. ниже описание символа "/").

4. Символ "/" (слеш). Служит для разделения классов в теге. Предположим, что в окне, которое нам нужно описать, имеется несколько уровней вложенности объектов, часть из которых нас не интересует, то есть работать с ними нам не нужно.
   Например, в диалоговом окне есть набор вкладок, на каждой вкладке определено еще одно диалоговое окно, невидимое для пользователя, но видимое для SilkTest-а, и уже в этом самом диалоговом окне находится кнопка "Старт", единственная, которая нам нужна. Если оставить фрейм в таком виде, как его записывает SilkTest, нам для нажатия на кнопку придется написать что-то в таком роде:

   **Code**

   ```
   Dialog1.PageList1.Dialog1.Start.Click ()
   ```

5. Однако писать такое каждый раз утомительно, а читать подобный код будет неудобно. Гораздо проще переместить кнопку "Старт" на один уровень с набором вкладок (используя сочетание клавиш Alt+Влево) и задать кнопке такой тег:
   ***tag "[PageList]#1/[DialogBox]#1/[PushButton]Старт"***
   После этого нажатие на кнопку будет выглядеть таким образом:

   **Code**

   ```
   Dialog1.Start.Click ()
   ```

6.
7. Символы ".." (две точки). Используются для указания предка данного объекта. Проще всего пояснить на примере.
   Есть окно, у которого постоянно меняется Window ID, местоположение и заголовок, однако у него гарантированно всегда есть кнопка "Print".
   Тег для этого окна будет таким:
   ***tag "[PushButton]Print/.."***
   Обратите внимание: это НЕ тег кнопки, а тег окна, который содержит такую кнопку.

   Другой пример: та же кнопка, но находится она всегда в диалоговом окне, который находится на наборе вкладок, а набор вкладок находится в нашем окне, для

которого нужен тег.
Он описывается так:
***tag "[PageList]#1/[DialogBox]#1/[PushButton]Смарт/../../.."***

8. "~ActiveApp". Указывает, что предок самого высокого уровня данного объекта в настоящий момент активен.
   Такой тег используется, например, для описания MessageBox, так как его непосредственный предок может быть любым (сравните предков для этого окна в TestApplication в двух случаях: когда оно вызывается через меню *Menu - The Item*, предком является главное окно приложения, когда же оно вызывается из любого диалогового окна, например Check Box, нажатием кнопки *Popup*, то предком будет это диалоговое окно).
   Также такой префикс необходимо указывать для диалоговых окон, которые не имеют главного окна (например, окна при установке некоторых программ: Microsoft Office, Acrobat Reader).
   Пример тега для таких программ:
   ***tag "~ActiveApp/[DialogBox]Установка Microsoft Office*"***

9. Номера объектов с одинаковыми тегами.
   Указывается в конце тега, заключается в квадратные скобки.
   Например, если у нас открыто несколько окон TestApplication, то при попытке обратиться к нему возникнет исключение
   ***** Error: Window '[MainWin]Test Application' is not unique**
   Однако можно исправить тег окна, чтобы такая ошибка не возникала:
   ***tag "Test Application[1]"***
   Чтобы теперь закрыть все окна TestApplication в тесткейсе, достаточно будет написать

   **Code**

```
[-] while wTestApp.bExists
        [ ] wTestApp.Close ()
```

10.
11. Символ "~" (тильда). Ставится в начале мультитега. Указывает SilkTest-у, что надо проверять и вторую часть мультитега (по умолчанию если первая часть совпадает, то вторая часть не проверяется).
    Пример:
    ***tag "~Test Application|$C:\Program Files\Segue\SilkTest\testapp.exe"***
    Если изменить в этом теге его вторую часть, то SilkTest не увидит приложение.
    Если после этого убрать тильду в начале тега, то приложение снова будет доступно.

Еще некоторые замечания по тегам.

В тегах можно использовать вызовы функций. Например, если у какого-то окна заголовок представляет собой текущую дату в формате **"Сегодня дд/мм/гггг"** (дд, мм, гггг - день, месяц и год), то целесообразно для этого окна указать такой тег:
***tag "Сегодня {FormatDateTime (GetDateTime (), "dd/mm/yyyy")}"***

Учитывая этот факт, а также то, что в классах SilkTest-а отсутствует понятие

конструктора и деструктора, можно добавить к тегу вызов функции, которая будет возвращать пустую строку, но при этом выполнять действия, которые должны выполняться в конструкторе:

**Code**

```
[-] string Constructor ()
        [ ] // Какие-то действия конструктора...
        [ ] return ""
[-] window MainWin wTestApp
        [ ] tag "{Constructor ()}Test Application[1]"
```

Однако при этом действия конструктора будут выполняться при каждом обращении к окну. Это можно использовать, например, в случае, если при каждом действии в окне изменяется его заголовок и это необходимо проверять.

### 1.2.3 Выделение общих винклассов

В случае небольших окон, скорее всего, никакой модификации не потребуется. Каждое окно описывается так, как описано выше, и на этом его описание заканчивается. Однако в случае больших приложений зачастую можно выделить общие элементы управления, диалоговые окна, меню и т.д., которые описываются одинаково, однако характерны для разных частей приложения.

Если внимательно посмотреть на окна, которые вызываются из меню Control приложения **Test Application**, можно выделить следующие общие элементы управления:

- кнопка **Exit**
- кнопка **Popup**
- чекбокс **Enabled**

Эти три элемента лучше выделить в отдельный винкласс, а затем все окна, содержащие эти элементы, пронаследовать от этого класса.
Для этого есть несколько причин:

1. мы избегаем повторяемости кода (описываем только один раз эти элементы)
2. получаем возможность написания методов для этого винкласса, которые будут доступны для всех окон
3. в случае изменения приложения (например, кнопка **Exit** изменит название на **Close**) нам будет достаточно сделать изменения один раз
4. улучшается тестирование (например, кнопка **Exit** изменится на **Close** во всех окнах, кроме одного; скорее всего, это будет неправильно)

Полученный класс будет иметь следующий вид:

**Code**

```
[-] winclass TestApp_Controls : DialogBox
        [ ] parent wTestApp
        [+] PushButton btnExit
                [ ] tag "Exit"
```

```
        [+] PushButton btnPopup
                [ ] tag "Popup"
        [+] CheckBox cbEnabled
                [ ] tag "Enabled"
```

Как видно, кроме объявления в винклассе элементов управления, в нем же объявлен предок: **parent wTestApp**. Теперь его не н ужно объявлять в каждом окне.
Теперь описания окон, наследующихся от этого класса, будут выглядеть так:

**Code**

```
window TestApp_Controls dCheckBox
```

и т.д.

Примечание. В SilkTest-е версий до 7.5 есть ошибка: при обращении к элементам управления, которые помещены в класс, как это сделано у нас, доступ к этим элементам не виден в выпадающем списке в окне редактора. Это может проявляться в некоторых случаях при большой вложенности классов.
В качестве примера посмотрите метод **CheckPopup()**, определенный в созданном винклассе **TestApp_Controls** (файл **TestApp.inc**). Этот метод проверяет, что при нажатии на кнопку **Popup**, появляется сообщение. Также посмотрите тесткейс **Test_Winclass_Method()** (файл **TestApp.t**)

## 1.3 Добавление свойств (property) и методов (method)

### 1.3.1 Свойства (properties)

Каждый раз, когда необходимо изменить данные в элементе управления (ввести текст в поле ввода, установить или снять флажок с чекбокса, выбрать элемент из списка и т.д.) имеется выбор между использованием стандартных свойств или методов. Например, для ввода текста в поле ввода можно воспользоваться двумя стандартными средствами этого элемента управления:

- использовать метод SetText ()
- использовать свойство sValue

Например, следующие две строки выполнят одно и то же: занесут заданную строку "Text" в поле ввода Field, которое находится в окне Dialog:

- Dialog.Field.SetText ("Text")
- Dialog.Field.sValue = "Text"

Кроме того, можно воспользоваться общим для всех окон методом TypeKeys:
**Dialog.Field.TypeKeys ("Text")**

Точно также можно работать и с другими элементами управления. Например, для элемента **RadioList** можно использовать метод **Select()** или одно из свойств: **sValue, iValue**.

На самом деле встроенными являются лишь методы, а свойства (**iValue, sValue** и т.д.) определены средствами SilkTest-а в файле **winclass.inc**.

Точно так же можно описывать свои свойства, которые могут пригодиться в работе.
У свойств могут быть 2 метода: **Set() и Get()**. Первый предназначен для установки значения, второй - для его (значения) извлечения.

Например, имеем следующее описание объекта в диалоговом окне:

**Code**

```
[-] window DialogBox dDialog
        [ ] tag "#1"
        [-] TextField edtField
                [ ] tag "#1"
```

Если мы хотим ввести какие-либо данные в это поле, мы можем воспользоваться одним из описанных выше способов. А можно немного сократить запись, используя свойства.
В качестве примера напишем свойство sField, при обращении к которому можно изменять и считывать значение из этого поля.
Выглядеть оно будет так:

**Code**

```
[-] property sField
        [-] STRING Get ()
                [ ] return this.edtField.sValue
        [-] VOID Set (STRING sValue)
                [ ] this.edtField.sValue = sValue
```

Поместить объявление этого свойства необходимо на одном уровне с полем edtField.
Теперь работа с этим полем немного упростилась:

- чтобы ввести значение, необходимо написать **dDialog.sField = "Text"** (сравните со стандартным **dDialog.edtField.sValue = "Text"** или **dDialog.edtField.SetText("Text")** )
- чтобы извлечь значение, необходимо написать **STRING sData = dDialog.sField** (сравните со стандартным **STRING sData = dDialog.edtField.sValue** или **STRING sData = dDialog.edtField.GetText()** ).

Это очень простой пример, однако могут встречаться и случаи сложнее.
Пример из практики: в приложении было текстовое поле с несколькими символами в нем, каждый из которых обозначал какую-то настройку. Каждый из символов влиял на функциональность всего приложения. Извлечь их было очень просто, стандартными методами SilkTest-а, а вот записать строку в это поле невозможно: строка была доступна только для чтения, а для изменения строки в этом поле служило диалоговое окно с кучей CheckBox'ов в нем. Если CheckBox включен - соответствующий символ появляется в строке. Это дополнительное окно открывалось по нажатию на кнопку рядом с полем ввода. Естественно, что каждый раз писать код для открытия этого окна и изменения значений CheckBox'ов не имеет смысла. Поэтому было описано новое свойство. Метод **Get()** для него был простой: использовался стандартный метод **GetText()**; а вот метод **Set()** был посложнее: он принимал строку символов, нажимал на кнопку, в диалоговом окне заполнял требуемые данные, после чего закрывал окно. Написать этот метод **Set()** было ненамного сложнее, чем написать эти же действия для какого-то конкретного случая, а экономия времени в дальнейшем велика: вместо кучи действий использовалось всего одно присваивание.

В качестве примера посмотрите на свойство **lsListView** в окне **dListView** (файл **TestApp.inc**) и тесткейс **Using_Properties** (файл **TestApp.t**).

1. Попробуйте расширить это свойство, чтобы вместо того, чтобы создавать колонку с заголовком **Column**, метод **Set()** принимал бы и имя колонки (подсказка: так как параметр у метода **Set()** может быть лишь один, воспользуйтесь созданием нового типа). Если на данном этапе задание кажется вам сложным - попробуйте вернуться к нему позже.
2. Придумайте и напишите подобное свойство для другого окна.

### 1.3.2 Аттрибуты (attribute)

Аттрибуты ассоциируются с оконным классом (НЕ окном!) и используются методами **GetEverything** и **VerifyEverything** для получения и проверки информации в окне. Для стандартных классов аттрибуты определены в файле **winclass.inc.** Кроме того, можно добавлять аттрибуты, определяемые пользователем.

Общий вид объявления аттрибута:

**Code**

```
attribute sAttribute, VerifyFunc, GetFunc
```

где:

- **sAttribute** - имя аттрибута
- **VerifyFunc** - имя функции, которая используется для проверки значения этого аттрибута
- **GetFunc** - имя функции, используемой для получения значения аттрибута

Например, для объявления нового аттрибута, который проверяет количество дочерних объектов в окне, необходимо написать следующее:

**Code**

```
        [ ] attribute "Number of children", VerifyNumChild, GetNumChild
        [-] INTEGER GetNumChild()
                [ ] return ListCount (this.GetChildren ()) // return count of
children of dialog
        [-] hidecalls VerifyNumChild (integer iExpectedNum)
                [ ] Verify (this.GetNumChild (), iExpectedNum, "Child number
test")
```

В первой строке мы определяем имя аттрибута и имена функций для проверки и получения значения аттрибута. Во второй строке мы описываем функцию для определения количества дочерних объектов. В четвертой строке мы описываем функцию, которая осуществляет проверку аттрибута с проверяемым значением (iExpectedNum) и генерирует исключение, если проверка не прошла успешно.

В файле TestApp.t есть тесткейс Using_Attributes, который демонстрирует использование аттрибутов для окна dCheckBox.

Примечание: аттрибуты считаются устаревшими и их не рекомендуется использовать. Вместо них рекомендуется использовать свойства (properties).

### 1.3.3 Методы (Methods)

Метод - это функция, определенная для конкретного окна или винкласса. Общий вид описания метода выглядит так:

**Code**

```
[gui-specifier] [return-type] method-id ( [arguments])
```

**gui-specifier** - определяет область видимости (private или public) метода
**return-type** - значение, возвращаемое методом; если не указано, то метод не возвращает никакого значения
**method-id** - имя метода
**arguments** - аргументы, передаваемые методу

Имя метода должно быть уникальным для данного класса и/или окна. Если у вас объявлен винкласс и окно этого класса, то методы можно объявлять как внутри класса, так и внутри окна. Однако желательно придерживаться какого-то одного стиля во избежание путаницы и упрощения поиска.

При объявлении метода в окне или винклассе этот метод будет доступен не только всем экземплярам этого класса, но и всем объектам внутри этого класса или окна, причем любой вложенности.

Например, для окна **wTestApp** определен метод **CloseAll()**, закрывающий все окна этого приложения, кроме главного. Вызов этого метода будет выглядеть так:

**Code**

```
wTestApp.CloseAll()
```

Однако, учитывая вышесказанное, мы можем написать следующее:

**Code**

```
wTestApp.File.New.CloseAll()
```

Такой код отработает нормально в том случае, если внутри нет обращений к каким-либо объектам главного окна, которые не определены для пункта меню *File - New* (например, при попытке таким же образом вызвать метод **AddMenu()** возникнет ошибка, так как данный метод использует элемент **MenuBar**. который для пункта меню не определен). SilkTest, начиная с версии 7.0, в подобных случаях при компиляции выдает предупреждение:
***** Warning: Function CloseAll is not defined for window wTestApp.File.New**

Однако все равно отрабатывает. Естественно, что делать так категорически не рекомендуется.

**Виртуальные методы**

В некоторых случаях вам может потребоваться объявить метод в классе, который потом будет переопределяться его потомками. Такой метод называется виртуальным (аналогичен виртуальным методам в c++). Для объявления виртуального метода необходимо в самом начале его объявления поставить ключевое слово **virtual**:

**Code**

```
virtual void MyMethod ()
```

После чего его можно переопределять в дочерних классах.

Если необходимо обратиться к исходному методу, который был переопределен, используется ключевое слово **derived::**

**Code**

```
derived::MyMethod()
```

В данном случае будет вызван метод предка.

В качестве примера рассмотрите тесткейс **Virtual_Methods_Demonstration** из файла **TestApp.t** и переопределенные методы окон, которые в нем используются, из файла **TestApp.inc**.
В этом примере рассмотрены следующие возможности:

1. Для класса **TestApp_Controls** (общего класса для практически всех окон, которые вызываются из меню *Control*) переопределен стандартный метод **Close()**. Он переопределен таким образом, чтобы для закрытия окна вызывать стандартный метод **Close()**, если параметр **bUseDerived** установлен в **TRUE**, или нажимать на кнопку **Exit** в остальных случаях (когда параметр **bUseDerived** не установлен или равен **FALSE**). Первый случай вызывается в примере 1, второй случай - в примере 4.
2. Для окна **dListView** метод **Close()** снова переопределен таким образом, чтобы нажимать комбинацию клавиш **Alt-E** для закрытия окна. Этот метод используется в примере 2.
3. В примере 3 для окна **dPageList** определен таким образом, чтобы вызывать метод предка (derived). Так как это окно наследуется от класса **TestApp_Controls**, соответсвенно вызывается метод этого класса.

Кроме того здесь показан пример, как использовать опциональные (необязательные) аргументы в функциях. Конструкция, которая используется в методе **Close()** винкласса:

**Code**

```
[-] virtual void Close (BOOLEAN bUseDerived NULL optional)
        [-] if (IsNull (bUseDerived) || !bUseDerived)
```

```
            [ ] Print ("Pressing button Exit to close dialog")
            [ ] this.btnExit.Click ()
        [-] else
            [ ] Print ("Using standard Close() method")
            [ ] derived::Close ()
```

В первой строке при указании параметров мы говорим, что это опциональный параметр (ключевое слово **optional**) и может быть равен **NULL**. Затем в самом методе делаем проверку: Если параметр не передан или передан **NULL** (IsNull (bUseDerived)) либо ( || ) равен **FALSE** (!bUseDerived), то выполняется первая часть: нажатие на кнопку. Если ни одно из этих условий не выполнилось (то есть передан параметр **TRUE**), то вызывается изначальный метод **Close().**

В качестве самостоятельного задания рекомендуется следующее: переопределите для главного окна **wTestApp** метод **Close()** таким образом, чтобы он сначала вызывал метод **CloseAll()**, а затем закрывал само окно с помощью вызова метода предка.

## 1.4 Использование Class Map и расширение встроенных классов

### 1.4.1 Использование Class Map

**Class Map (Карта Классов)** - это весьма полезный инструмент, который позволяет указать SilkTest-у, как работать с нестандартными классами в том случае, если эти классы соответствуют стандартным, но называются иначе. Особенно часто такая ситуация встречается в случае **Delphi**-приложений, в которых практически все классы определяются SilkTest-ом как нестандартные.

Чтобы это сделать откроем **Class Map** (*Options - Class Map* или *Record - Window Declaration - Class Map*), в поле **Custom Class** введем имя класса, который нам необходимо описать как стандартный, а в выпадающем списке справа Standard Class выберем стандартный класс, к которому мы хотим присоединить наш нестандартный класс.

Например, если на форме имеется элемент **ListBox**, то SilkTest будет его определять как **CustomWin TListBox** и, как следствие, работать с ним будет невозможно. Однако если в **Class Map'e** определить этот класс как ListBox, то все операции, которые можно производить с обычным **ListView**, станут доступны и для класса **TListView**. Однако так поступить можно не со всеми элементами (например, с классом **GridControl** это не поможет).

Обратите внимание: делать описание окон с помощью **Record Window Declaration** надо после того, как нестандартный класс примаплен к стандартному. Для проверки того, что вы сделали все правильно, откройте выберите пункт меню *Record - Window Declaration*, наведите курсор мыши на примапленный элемент и убедитесь, что в окне record **Window Declaration** его класс определяется не как **CustomWin**, а как класс, к которому вы его примапили. После этого можно записывать объявления окон.

Отдельного внимания заслуживает пункт **Ignore** в списке стандартных классов **Class Map'а**. Этот пункт предназначен для игнорирования какого-либо класса, или группы классов (в именах **Custom Class'ов** можно использовать символы групповой замены). После добавления такого соответствия добавленный класс будет проигнорирован и все элементы данного класса не попадут в описание окна.

В этом случае элементы данного класса не будут отображаться в списке элементов окна **Window Declaration**, однако их отображение можно включить, если в этом окне нажать кнопку Options, а затем включить опцию "Show ignored windows". При включенной этой опции в колонке **Identifier** окна **Window Declaration** будет отображаться значение **(Ignored)**.

Использовать данную возможность следует осторожно, так как можно "заигнорить" какой-нибудь важный класс. Если это просто элементы окна, то лучше оставить их как есть, а если элементы этого класса встречаются в середине иерархии объектов, то лучше воспользоваться корректировкой тегов и переместить все дочерние объекты на уровень вверх (подробнее об этом в главе "1.2.2 Работа с тегами (tags)").

### 1.4.2 Расширение встроенных классов

Уже описанные в SilkTest-е стандартные классы имеют большое количество методов и свойств, которые позволяют работать с элементами управления. Однако, возможно, вам захочется иметь еще какие-то методы и свойства, доступные уже имеющимся классам. В этом случае вы можете выбрать один из способов:

- переопределить стандартный класс
- объявить новый класс, пронаследовав его от стандартного, а затем объявлять во фрейме объекты этого класса вместо стандартного

Конечно, можно добавлять эти методы и свойства в само определение класса (файл **Program Files\Segue\SilkTest\winclass.inc**), однако делать это не рекомендуется, так как можно потерять сделанные изменения во время переустановки SilkTest-а. Кроме того, эти изменения не будут доступны при переносе на другой компьютер.
Поэтому лучше создать отдельный **inc**-файл, в который заносить все подобные изменения и подключать его по мере необходимости.

В качестве примера рассмотрим стандартный класс **ListView** и его метод **GetItemText**, который возвращает значение ячейки по номеру строки, а также имеет необязательный параметр **iColumn**, который позволяет указать номер колонки, из которой извлекать текст. Было бы удобнее, если бы в качестве второго параметра передавался бы не номер колонки, а ее заголовок (особенно в случае, когда пользователь может настраивать расположение колонок).

Можно переопределить имеющийся метод **GetItemText** (как это сделать описано подробнее в главе "1.3.2 Методы (Methods)"), однако данный пример иллюстрирует создание нового метода для имеющихся классов, поэтому мы создадим новый метод **GetItemTextByColumn**. Этот метод будет возвращать строку, находящуюся в нужной нам ячейке, либо **NULL** в случае, если такая колонка не найдена.

Для этого в файле **TestApp.inc**, в котором хранится вся функциональность нашего примера, мы создадим следующее объявление:

**Code**

```
[-] winclass ListView : ListView
```

Это в том случае, если вы хотите переопределить имеющийся класс. Если же вы хотите создать новый класс и пронаследовать его от **ListView**, то необходимо написать:

**Code**

```
[-] winclass ListViewExt : ListView
```

И после этого изменить тип объектов **ListView** на **ListViewExt**. Мы рассмотрим первый вариант.

Итак, теперь необходимо написать новый метод. Выглядеть он будет так:

**Code**

```
[+] winclass ListView : ListView
        [+] STRING GetItemTextByColumn (in INTEGER iRow, in STRING sColumn)
                [ ] INTEGER iColumn = 0
                [ ] INTEGER i
                [ ]
                [+] for i = 1 to this.GetColumnCount ()
                        [+] if (this.GetColumnName (i) == sColumn)
                                [ ] break
                [ ]
                [+] if (iColumn != 0)
                        [ ] return this.GetItemText (iRow, i)
                [+] else
                        [ ] return NULL
```

Теперь можно не считывать каждый раз все имена колонок с целью выяснить, какая из них нам нужна, достаточно использовать этот метод.

Для примера посмотрите тесткейс **Test_Overrided_Class** из файла **TestApp.t**, который использует новый метод. В качестве тренировки переопределите стандартный класс **TextField**. Например, в случае если в поле несколько строк, извлечь их можно с помощью метода **GetContents()**, который возвращает список строк (LIST OF STRING). Напишите метод **GetStringContent**, который будет вместо списка строк возвращать строку, в которой строки поля будут разделены точкой с запятой (к примеру, если в поле введены три строки "раз", "два", "три", то метод должен вернуть строку вида "раз;два;три").

# 3. Recovery-система

При разработке часто может возникнуть необходимость написания кода, который позволит подготовить тестовую среду к запуску скриптов или вообще выполнить некоторые предварительные действия. Например, возникает необходимость выставить определенные настройки SilkTest-а исходя из конфигурации конкретной машины, на которой нужно провести тестинг. Настройки могут зависеть от операционной системы, языка системы и т.д. Бывают также случаи, когда нужно проинициализировать некоторые глобальные переменные. Зачастую эти величины нужно считывать из какого-то файла в определенном формате. В качестве примера можно привести величину, означающую количество секунд, с течение которых ожидается появление некоторого окна. Естественно, можно завести подобную переменную и задать ей некоторое значение явно. Но в разных средах тестируемое приложение работает с разными скоростями. Где-то приложение работает быстро, а где-то очень медленно. Соответственно на ожидание окна потребуется разное время. Как следствие - переменная, хранящая это время ожидания может быть инициализирована по-разному. Но постоянно править файл, в котором объявлена данная переменная, не очень хорошо. Более того, значения подобных варьируемых величин лучше выносить в некоторый отдельный файл и все настройки осуществляеются только в этом файле.
Теперь нужно обеспечить инициализацию таких данных перед запуском скриптов. Одним из вариантов является разработка функции или тесткейса, который осуществит необходимую инициализацию и запустить его перед запуском других тесткейсов. Но у этого варианта есть недостатки. Каждый тесткейс может запускаться как в группе ( тестпланом например ) так и поодиночке. Во втором случае инициализацию данных придется интегрировать в каждый тесткейс, что не очень хорошо. Более удобным вариантом будет помещение инициализации данных в **appstate**. Соответственно, данные будут готовы до того, как тесткейс начнет выполнение. Данный механизм вполне эффективен, если **appstate** всего один. Но если их несколько и они не пересекаются по выполняемым действиям, то этот подход уже хуже работает. Более того, **appstate** вызывается столько раз, сколько вызывается тесткейсов, хотя зачастую инициализацию достаточно произвести один раз при запуске всей группы тесткейсов, находящихся в одном файле ( например при нажатии F9 в файле, содержащем только тесткейсы ). Плюс ко всему **appstate** может сам использовать некоторые величины, которые уже должны быть проинициализированы.
В любом случае, нужно реализовать некоторую функцию, которая сработает до того, как какой-либо тесткейс вообще начнет работу. Для таких целей в SilkTest-е существует так называемая **Recovery**-система. Данная система реализована в виде набора функций:

- ScriptEnter - выполняется в самом начале выполнения файла скрипта
- ScriptExit - выполняется сразу по завершении выполнения файла скрипта
- TestCaseEnter - выполняется сразу перед началом выполнения отдельного тесткейса (до **appstate**)
- TestCaseExit - выполняется сразу после завершения тесткейса ( после **appstate**)
- TestPlanEnter - выполняется сразу перед началом выполнения тестплана
- TestPlanExit - выполняется сразу после выполнения тестплана

Это набор функций, который нужно переопределить. Если какая-то из этих функций не переопределена, то задействуются функции по-умолчанию (DefaultScriptEnter, DefaultScriptExit, DefaultTestCaseEnter, DefaultTestCaseExit, DefaultTestPlanEnter, DefaultTestPlanExit).

В начале данной главы **Recovery**-система преподносилась как средство для выполнения некоторого кода до начала выполнения тесткейса. Но как видно из вышеприведенного перечня функций данной системы, также возможно выполнение действий и после выполнения отдельного тесткейса или целого скрипта. Особенностью Exit-функций данной системы является то, что они принимают параметром BOOLEAN-значение, которое позволяет установить, завершилась ли работа скрипта, тесткейса, тестплана в результате генерации исключения. Это достаточно полезно, при генерации исключения в скриптах ( при условии, что исключение не перехватывается ) происходит просто обрыв выполнения без надлежащей выдачи информации. А **Recovery**-система подобную ситуацию может отследить и предоставляет возможность выдать необходимую информацию.
Рассмотрим действие данной системы. Определим функции **Recovery**-системы так, чтобы они выдавали в отчет результаты своей работы. Примерно так:

**Code**

```
[+] ScriptEnter()
        [ ] Print( "Script Enter" )
[ ]
[+] ScriptExit( BOOLEAN bException )
        [+] if( bException )
                [ ] LogError( "Script Exit by Exception" )
                [ ] ExceptPrint()
                [ ] raise 1,"Exception in Script Exit"
        [+] else
                [ ] Print( "Script Exit" )
[ ]
[+] TestCaseEnter()
        [ ] Print( "Testcase Enter" )
        [ ] DefaultTestCaseEnter()
[ ]
[+] TestCaseExit( BOOLEAN bException )
        [+] if( bException )
                [ ] LogError( "TestCase Exit by Exception" )
                [ ] ExceptPrint()
        [+] else
                [ ] Print( "TestCase Exit" )
        [ ] DefaultTestCaseExit(bException)
[ ]
[+] TestPlanEnter()
        [ ] Print("TestPlan Enter")
[ ]
[+] TestPlanExit( BOOLEAN bException )
        [+] if( bException )
                [ ] LogError( "TestPlan Exit by Exception" )
                [ ] ExceptPrint()
        [+] else
                [ ] Print( "TestPlan Exit" )
```

В данном случае функции **Recovery**-системы просто выдают сообщение о том, что та или иная функция вызвана. Поместим этот код в некоторый файл, напрмер **Recovery.inc**.
Создадим некоторый тестовый t-файл (например **RecTest.t**) , в котором напишем 2 тесткейса: первый пройдет успешно, а второй сгенерирует исключение. Файл имеет вид:

**Code**

```
[ ] use "Recovery.inc"
[ ]
[+] appstate TestAppstate() basedon none
        [ ] Print("Appstate. Testcase state: {GetTestcaseState()}")
[ ]
[+] testcase UsualTest() appstate TestAppstate
        [ ] Print("Usual Test running...")
[ ]
[+] testcase ExceptionTest() appstate TestAppstate
        [ ] raise 0,"Test exception"
```

Таким образом мы сейчас можем проверить очередность вызовов функций **Recovery**-системы, **appstate**-ов и тесткейсов. Запустим тесткейс **UsualTest** отдельно ( выбираем меню **Run > Testcase**, в появившемся окне выберем тесткейс **UsualTest** и нажмем кнопку **Run**). После выполнения, файл результатов будет содержать следующие строки:

**Code**

```
[ ] Script Enter
[+] Testcase UsualTest - Passed
        [ ] Testcase Enter
        [ ] Appstate. Testcase state: TCS_ENTERING
        [ ] Usual Test running...
        [ ] TestCase Exit
        [ ] Appstate. Testcase state: TCS_EXITING
[ ] Script Exit
```

Итак, из результатов видно, что последовательность вызовов такова:

1. ScriptEnter
2. TestCaseEnter
3. TestAppstate
4. UsualTest
5. TestCaseExit
6. TestAppstate
7. ScriptExit

Во 2-й главе говорилось, что **appstate** вызывается сразу до и сразу после выполнения тесткейса, но результаты последнего запуска показывают, что **TestCaseExit** выполняется до **TestAppstate**. Поэтому нужно разобраться, как же **appstate** взаимодействует с **Recovery**-системой. В файле **Recovery.inc** закомментируем вызовы **DefaultTestCaseEnter()** и **DefaultTestCaseExit(bException)**. Повторим предыдущий запуск. На этот раз результаты имеют вид:

**Code**

```
[ ] Script Enter
[+] Testcase UsualTest - Passed
        [ ] Testcase Enter
        [ ] Usual Test running...
        [ ] TestCase Exit
[ ] Script Exit
```

То же самое, только **appstate** не выполнился. Получается, что в Default-функциях **Recovery**-системы как раз и обеспечивается запуск **appstate**. Вы можете в этом убедиться, если посмотрите файл **defaults.inc** в директории, в которой находится SilkTest. В данном

файле как раз и определены функции **DefaultTestCaseEnter()** и **DefaultTestCaseExit(bException)**. Таким образом **appstate** запускается при помощи **Recovery**-системы. Можно, конечно, и обойти это путем вызова функции **SetAppState()** в функциях **TestCaseEnter** и **TestCaseExit**, но все-таки предпочтительнее пользоваться стандартными механизмами.
Раскомментируем вызовы **Default**-функций в **Recovery.inc** и рассмотрим поведение **Recovery**-системы при других видах запуска скриптов ( помимо индивидуального ). Итак, перейдем к файлу **Test.t** и запустим все тесткейсы с помощью клавиши **F9**. Вывод будет иметь вид:

**Code**

```
[ ] Script Enter
[+] Testcase UsualTest - Passed
        [ ] Testcase Enter
        [ ] Appstate. Testcase state: TCS_ENTERING
        [ ] Usual Test running...
        [ ] TestCase Exit
        [ ] Appstate. Testcase state: TCS_EXITING
[+] Testcase ExceptionTest - 2 errors
        [ ] Testcase Enter
        [ ] Appstate. Testcase state: TCS_ENTERING
        [ ] TestCase Exit by Exception
        [ ] Test exception
        [ ] Occurred in ExceptionTest at Test.t(575)
        [ ] Test exception
        [ ] Occurred in ExceptionTest at Test.t(575)
        [ ] Appstate. Testcase state: TCS_EXITING
[ ] Script Exit
```

Как видно, ScriptEnter/Exit-функции вызвались по разу. Также функцией **TestCaseExit** было перехвачено исключение, сгенерированное в тесткейсе **ExceptionTest**. Аналогичные результаты будут получены, если эти оба тесткейса будут вызываться функцией **main().**
И последний способ запуска, который будет рассмотрен - это запуск тесткейсов в тестплане. А в тестплан можно включить не только тесткейс-функцию, но и обычную функцию. В файле **Test.t** допишем тестовую функцию:

**Code**

```
[+] VOID FunctionException()
        [ ] raise 0,"Function Exception"
```

Эта функция будет генерировать исключение, что позволит нам отследить, какая функция **Recovery**-системы это исключение перехватит. Создадим файл тестплана **Test.pln** со следующим содержимым:

**Code**

```
[+] script: Test.t
        [+] 1
                [ ] testcase: UsualTest()
        [+] 3
                [ ] testcase: FunctionException()
        [+] 2
                [ ] testcase: ExceptionTest()
```

Теперь этот файл можно запустить нажатием на **F9**. Результаты будут иметь вид:

**Code**

```
[+] 1
        [ ] TestPlan Enter
        [ ] Script Enter
        [ ] Testcase Enter
        [ ] Appstate. Testcase state: TCS_ENTERING
        [ ] Usual Test running...
        [ ] TestCase Exit
        [ ] Appstate. Testcase state: TCS_EXITING
        [ ] Script Exit
[+] 3
        [ ] Script Enter
        [ ] Script Exit by Exception
        [ ] Function Exception
        [ ] Occurred in FunctionException at Test.t(578)
        [ ] Exception in Script Exit
        [ ] Occurred in ScriptExit at Test.t(542)
[+] 2
        [ ] Script Enter
        [ ] Testcase Enter
        [ ] Appstate. Testcase state: TCS_ENTERING
        [ ] TestCase Exit by Exception
        [ ] Test exception
        [ ] Occurred in ExceptionTest at Test.t(575)
        [ ] Test exception
        [ ] Occurred in ExceptionTest at Test.t(575)
        [ ] Appstate. Testcase state: TCS_EXITING
        [ ] Script Exit
        [ ] TestPlan Exit
```

Как видно из результатов, функции **TestPlanEnter/Exit** вызываются по разу в самом начале и в самом конце работы тестплана соответственно. Вызовы функций **TestCaseEnter/Exit** и **ScriptEnter/Exit** осуществляются парно для тесткейсов, за исключением запуска функции **FunctionException**, которая тесткейсом не является. Для последней, исключение перехватывается в функции **ScriptExit**. В этом состоит очередное отличие обычных VOID-функций от тесткейсов: тесткейсы имеют стандартную привязку к **Recovery**-системе посредством функций **TestCaseEnter** и **TestCaseExit**.
Подведем итог: **Recovery**-система - это система функций, которые позволяют проводить подготовку среды к тестированию, подготавливают тестируемое приложение и приводят его в некоторое начальное состояние до начала выполнения тесткейсов, а также восстанавливают настройки, параметры и восстанавливают начальное состояние тестируемого приложения после выполнения тесткейсов. При этом данные действия могут выполняться до и после начала работы тесткейса, захода в файл или работы тестплана, что позволяет подготавливать среду на разных этапах работы скриптов.

# 4. Работа с веб-приложениями

На данный момент число програмных продуктов, основанных на web, растет с каждым днем. Причем в данном случае подразумеваются не разовые проекты, а достаточно сложные приложения, которые разрабатываются в течение долгого периода времени и требуют достаточно обширного тестирования, в том числе и при переходе от версии к версии - то есть те приложения, тестирование которых имеет смысл автоматизировать. Рассмотрим особенности работы с веб-приложениями. Во-первых, веб-приложения работают не со стандартными GUI-объектами, а с классами, соответствующими модели DOM. Соответственно, это требует подключения соответствующего расширения. Точнее, его надо активировать. Для этого:

- Открываем броузер, под который будут разрабатываться скрипты, открываем некоторую страницу
- открываем **SilkTest**
- выбираем **Tools > Enable Extensions**
- В появившемся окне выбираем нужное приложение, жмем **Select**
- Далее появится диалог настройки расширений броузера. Выбираем пункт **DOM**, жмем **OK**
- Появится окно с сообщением, что активация расширений выполнена успешно.
- Закрываем окно

Эта процедура проделывается один раз. Если у вас возникли трудности с подключением расширения, обратитесь к главам 8. Использование расширений и 8.4 Расширения Explorer'a, в которых рассказывается, как подключить расширения вручную.
Итак, когда расширения уже подключены, можно рассмотреть оконные классы для веб-приложений. Их относительно немного (около 20), более того, в основном в используется не более десятка классов (это как правило). То есть количество элементов на самом деле не такое уж и большое. Но от этого особо легче не становится. Сфера применения объектов таких видов, как рисунок или таблица, может быть очень большой. Например рисунок может содержать просто некоторое изображение (обычная картинка), это может быть ссылка, это вообще объект, нажатие на который приводит к выполнению некоторого куска кода. С таблицами еще хитрее. Это могут быть списки данных, то есть таблицы в их традиционном понимании (как статические ,так и динамические). Также в веб-страницах таблицы используются для расположения элементов на форме. В данном случае таблица выступает только средством выравнивания. Затем, можно вспомнить выпадающие пункты меню, причем не стандартное меню, а сделанное при помощи скриптов. Такие меню реализуются тоже как таблицы, каждый элемент которой - ссылка на некоторую страницу. Как видно, применяются эти элементы по-разному, соответственно и работать с ними надо по-разному. Но все по порядку. Фрейм тоже имеет свои особенности. Во-первых, для всех веб-приложений есть только одно главное окно. Это окно **Browser**. Это универсальное окно броузера. Есть конечно другие окна, которые соответствуют уже конкретным броузерам (IE, Netscape), но это окно является универсальным для них. Во-вторых, все страницы записываются как объекты **BrowserChild**, у которых родительским окном является **Browser**. Это вызывает целый ряд трудностей, но все они в целом связаны с тем, что одновременно может быть открыто несколько окон броузера и нужно что-то сделать на той странице, которая в данный момент не является активной. Активировать ее не так-то и просто. **BrowserChild**-окна наследуют функциональность **ChildWin**-окон и работать с ними можно так же, как и с дочерним окном документа в однодокументном приложении. Самая большая трудность заключается в том, что без дополнительных настроек или дополнительной функциональности этим окнам нельзя

сделать **SetActive**. Такая операция вызовет исключение.
В-третьих, существует возможность регулировать распознавание объектов, то есть указать, какие объекты распознать, а какие нет. Это относится к объектам типа **HtmlMeta**,**HtmlHidden**,**XMLNode**,**HtmlText**,**HtmlForm**. Это связано с тем, что **SilkTest** обрабатывает HTML код и если распознавать все имеющиеся объекты, то их получится очень много и соответственно Агенту потребуется больше времени для нахождения нужных элементов. Также, это полезно тем, кто получает первоначальный фрейм путем записи деклараций окон. В этом случае сгенерируется очень много объектов, среди которых будет очень много "мусора". Например, объекты **HtmlMeta**,**HtmlHidden** редко учавствуют в скрипте и как правило они не нужны. Поэтому, значительно проще запретить распознавание таких объектов, чем очищать их из месива сгенерированного кода. Регулируется уровень распознавания объектов следующим образом:

- Выбираем меню **Options > Extensions**
- В появившемся диалоге выбираем броузер, для которого мы делаем настройки и нажимаем на кнопку **Extensions...**
- В появившемся диалоге настроек устанавливаем/сбрасываем флажки для тех элементов, с которыми нам нужно работать или не нужно работать.

Здесь это все настраивается. Не будем спешить с закрытием диалога настроек, так как есть еще одна особенность веб-приложений, которая в принципе попадает под данный пункт, но эта особенность вызывает достаточно большое количество проблем ( и вопросов по их решению ). Поэтому рассмотрение данного вопроса вынесено отдельно. Итак ...
В-четвертых, существует возможность регулирования уровня распознавания таблиц с нулевой шириной линий. Таблицы в HTML, как уже говорилось, могут использоваться по-разному - и как средство наглядного представления/группировки данных и как средство форматного расположения элементов ( например элементов управления на форме). Таблицы могут быть вложенными. Также следует отметить тот факт, что если в HTML с таблицами работают построчно (то есть описывается таблица построчно), то SilkTest разбивает таблицу на колонки, для чего имеется отдельный класс **HtmlColumn**. Такой подход для SilkTest-а вполне оправдан, если данные действительно расположены в таблице со строками и столбцами, но если таблица используется только для форматирования и не представляет из себя регулярной сетки, то тогда описание фрейма будет представлять из себя достаточно страшное зрелище. Соответственно, нужно задать некоторый уровень распознавания, при котором таблицы видны только до определенного уровня вложенности. Для этих целей в диалоге настройки расширений ( который я просил не закрывать) есть ползунок в секции **Borderless Table**. В этой секции как раз и регулируется уровень распознавания таблиц с нулевой границей. В зависимости от этого уровня, таблицы распознаются по-разному. В нижеприведенной таблице указаны правила распознавания таблиц с нулевой толщиной границы:
Если рассматриваемая ячейка таблицы **не** содержит элементов ввода

| Содержит ли таблица вложенные таблицы | Содержит какая-либо из вложенных таблиц элементы ввода | Уровень вложенности таблиц | Значение уровня распознавания таблиц |
|---|---|---|---|
| Нет | - | - | $0 < x < 0.30$ |
| Да | Нет | 1 | $0.3 < x < 0.60$ |
| Да | Нет | 2 | $0.6 < x < 0.75$ |
| Да | Нет | > 2 | 1 |
| Да | Да | не имеет значения | 1 |

Если рассматриваемая ячейка таблицы содержит элементы ввода

| Содержит ли таблица вложенные таблицы | Содержит какая-либо из вложенных таблиц элементы ввода | Уровень вложенности таблиц | Значение уровня распознавания таблиц |
|---|---|---|---|
| Нет | - | - | 0.75 < x < 0.9 |
| Да | Нет | 1-2 | 0.75 < x < 0.9 |
| Да | Нет | 3 | 0.91 < x < 0.99 |
| Да | Нет | > 3 | 1 |
| Да | Да | не имеет значения | 1 |

Возможно, возникают трудности с разбором данных таблиц, но, как показывает практика, наиболее оптимальным является значение 0.76, поэтому, если мало желания разбирать вышеприведенные таблицы, то попробуйте устанавливать именно такое значение. В этом случае все, что надо, будет увидено.

## 4.1. Стратегия описания фрейма

Способов описания фрейма для веб-приложений, как для любого другого вида приложений, может быть столько, сколько тех людей, кто описывает эти фреймы. Но тем не менее, есть один общий принцип, которым можно руководствоваться. Большинство веб-приложений содержат некоторую часть, которая постоянна для всех страниц (или какой-то части) приложения. Это как правило шапка с главным меню и "footer". Как правило эти реквизиты и при разработке составляют каркасс страницы (то есть разработчики пишут некоторый шаблон, в который помещают все необходимое для той или иной страницы), а страница в понимании SilkTest-а является окном. Каркасс страницы - каркасс окна - базовая часть, характерная для определенной группы окон - **winclass.** Таким образом, практически каждое веб-приложение содержит в себе некоторый каркасс, который может быть отражен в **winclass**-е, возможно даже специфичном для данного приложения, но он имеет место. При описании фрейма имеет смысл выделить эту часть.

## 4.2. Тактика описания фрейма

### 4.2.1. Очистка фрейма от ненужных объектов

В процессе описания фрейма возникает множество мелких задач, которые приходится решать достаточно часто. Например, для веб-приложений характерно обилие элементов управления, которые не несут функциональности, необходимой для автоматизации, но фрейм засоряют тем не менее изрядно. Для веб-приложений это характерно особенно, так как на веб-страницах много элементов, задача которых просто заключается либо в форматировании, либо в каких-то других красивостях и функционального значения они не имеют. Реально необходимых элементов значительно меньше. Так, например, имеет смысл держать элементы, которые могут принимать данные, а также те элементы, нажатие мыши (или некоторой клавиши) на которой повлекут за собой срабатывание некоторой функциональности. Исключение могут составлять объекты заголовков или текстовое поле с динамическим содержимым ( текст может меняться). Заголовки могут быть использованы как объекты, уникально идентифицирующие некоторое окно. Для веб-приложений как раз наиболее характерна проблема в том, что сами объекты страниц описываются с одинаковыми тегами и их надо как-то различать между собой. Вот таким критерием различия и может выступать текст заголовка. А текстовые поля с динамическим содержимым отличаются тем, что они являются не только частью интерфейса, а еще и функциональности, что автоматически влечет необходимость использовать такие объекты во фрейме.

### 4.2.2. Динамические таблицы

Отдельно следует рассмотреть динамические таблицы. Такие таблицы достаточно описать до уровня колонок. Ниже не имеет смысла, так как дальше идет динамические элементы, которые как-то фиксированно и не описать. Более того, имеющаяся функциональность уже может получить нужную информацию из нужных ячеек. Так, например, в текстовых таблицах, в которых в каждой ячейке находится некоторый текст (неделимый с точки зрения хранения данных ), не имеет смысла декларировать объекты **HtmlText** как дочерние элементы соответствующего объекта класса **HtmlColumn**, так как класс **HtmlColumn** уже содержит в себе готовые методы по извлечению текста из нужной строки колонки. А если нужно нажать на текст в колонке или проделать еще какие-то действия с ним, то тут можно обойтись обращением к элементу через его оконный класс, например

**Code**

```
        [ ] wSomeBrowserChild.tblTable.clnNumber.HtmlText("#12").Click()
```

Если каждая ячейка колонки содержит только один объект текста, то такой код приводит к нажатию кнопки мыши на 12-м элементе соответствующей колонки. По аналогии можно действовать с колонками, содержащими любые другие элементы (главное, чтоб по одному на ячейку и одного и того же типа), например рисунки, кнопки и т.п. В этом случае можно даже создать отдельный оконный класс, который работает с колонками, содержашими в ячейках по элементу некоторого вида. Например:

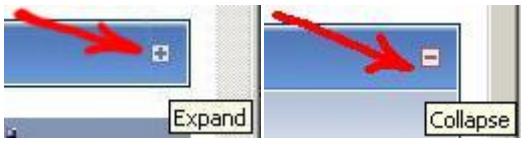
**Code**

```
[+] winclass ImageColumn : HtmlColumn
        [+] BOOLEAN ClickImage( INTEGER iRow, INTEGER iButton NULL optional,
INTEGER x  NULL optional, INTEGER y NULL optional, BOOLEAN bRawEvent  NULL
optional )
                [+] do
                        [
]this.HtmlImage("#{iRow}").Click(iButton,x,y,bRawEvent)
                [+] except
                        [ ] ExceptPrint( )
                        [ ] return FALSE
                [ ] return TRUE
```

Это был пример расширения для класса колонки рисунков, на каждом из которых можно сделать клик мышью. По аналогии можно расширить и для других элементов.

### 4.2.3. Нестандартная функциональность объектов

Как уже было указано, некоторые HTML-объекты могут применяться самыми различными способами, то есть представлять из себя некоторый функциональный элемент, имеющий свою специфику. Это заключается в возможности использовать в веб-страницах различные скрипты, как на серверной, так и на клиентской стороне. Наиболее интенсивно дополнительная функциональность задействуется в рисунках. В качестве примера приведу ссылку http://forums.software-testing.ru/index.php?showforum=79. В таблице тем для каждого из форумов есть элемент управления, позволяющий разворачивать/сворачивать список тем. У него есть 2 состояния, которые показаны на рисунках:



В данном случае не имеет смысл присваивать тег данному объекту, исходя из его Caption, так как эта составляющая варьируется. В этом случае имеет смысл использовать другие составляющие для тега ( например индекс ). Более того, если элемент данного вида встречается не в единичном экземпляре ( а в данном примере такой элемент не один ), то имеет смысл выделить его в отдельный оконный класс, в который можно добавить методы, позволяющие:

- Разворачивать/сворачивать список тем
- Проверять состояние данного объекта

Проверить текущее состояние данного элемента можно путем извлечения Caption и сравнения с некоторыми допустимыми значениями.
Схожая ситуация встречается в динамических таблицах, которые функционально повторяют объект класса **ListView**. В частности я подразумеваю наличие в заголовке колонки рисунка, который обозначает порядок сортировки элементов данной колонки.

### 4.2.4. Одно главное окно

Этот пункт изначально не планировалось включать, так как это скорее вопрос идеологии. Но вообще, для веб-приложений предусмотрено несколько окон, соответствующих окну браузера, в частности **Explorer** и **Netscape** для соответствующих браузеров. Они уже описаны достаточно детально и содержат в себе необходимую функциональность. Более того, имеется окно **Browser**, которое может являться окном одного из браузеров. Какого именно - определяется установкой опции **Default browser** в диалоге **Options > Runtime.**
Таким образом, изначально расчитано на то, что фрейм для веб-приложения будет содержать только одно окно класса **MainWin**, которое уже определено в файлах, поставляемых вместе с SilkTest-ом.

**Совет**

Для более детального ознакомления с уже имеющейся функциональностью, используемой при написании скриптов для тестирования веб-приложений можно посмотреть файлы, расположенные в папке **Extend** в каталоге, в который установлен SilkTest. В частности это файлы **explorer.inc , netscape.inc, browser.inc**. Помимо объявлений основных окон там имеются объявления различных вспомагательных окон и стандартных модальных диалогов. Поэтому, прежде чем приступать к написанию фрейма, желательно хотя бы изучить тот фрейм, который уже был приготовлен, а затем уже приступать к написанию фрейма, специфического для конкретного приложения

Данный момент был затронут неспроста, поскольку в разных случаях люди описывают главные окна браузеров. Это могут быть те, кто только начинает осваивать SilkTest и еще плохо ориентируются в данной среде, в имеющихся возможностях. Соответственно такие люди записывают во фрейм все окна. Также возможен вариант, когда человек перешел к написанию скриптов для веб-приложений после работы, например, с GUI-приложениями. Соответственно выработанный подход к описанию фрейма начинает применяться и для веб-приложений.
По большому счету это нельзя назвать ошибкой, так как:

- От этого скрипты не станут работать хуже
- Наличие объектов **MainWin** во фрейме соответствует реальному строению окон веб-приложений
- Классический способ активации нужного окна ( метод **SetActive** )

Но тем не менее, описание таких окон занимает место во фрейме и, самое главное, это занимает время, обилием которого мало кто может похвастаться. Исходя из этих

соображений уже нужно решать, а имеет ли смысл многократно переписывать тот код, который уже написан и готов к использованию.
В данном подпункте я осветил принцип использования окон, который я считаю наиболее рациональным. Но есть один маленький нюанс: возникают проблемы, если открыто несколько окон браузера и нужно активизировать некоторую неактивную в данный момент страницу. Выражение **Browser.SetActive()** в данном случае не подойдет, так как окно **Browser** соответствует последнему активизированному окну браузера и в общем случае нужная страница не будет активизирована. В классе **BrowserChild**, а точнее в его базовом классе **BrowserChildRoot** определен метод **SetActive**. Но не везде этот метод работает корректно. Зачастую вызов данного метода сопровождается генерацией исключения. В таких случаях удобно использовать некоторое единое решение. Например, данный метод можно описать так:

**Code**

```
[+] winclass BrowserChild : BrowserChild
        ..........................................................
        [+] hidecalls SetActive()
                [ ] this.GetParent().SetActive()
```

То есть подобным образом можно переопределить класс веб-страницы. Данный метод сработает, если объект класса **BrowserChild**, у которого вызывается метод **SetActive**, является непосредственным дочерним окном главного окна браузера ( в иерархии нет промежуточных звеньев ). В противном случае нужно выбираться по иерархии вверх, пока не будет достигнуто главное окно или Desktop - самое верхнее окно в иерархии. Во втором случае главное окно браузера считается ненайденным.

## 4.3. Практика описания фрейма

От общих идей перейдем к конкретным задачам. В качестве примера возьмем страничку www.google.com, введем некоторое ключевое слово поиска, например **TEST**, и опишем окно результатов поиска. То есть у нас будет относительно бесформенный список ссылок, помещенных в определенные формы. Данный пример является достаточно показательным, так как он демонстрирует общие принципы описания фрейма для веб-приложений, но самое главное - данный пример покажет, как можно группировать элементы так, чтобы потом к ним можно было обращаться динамически. А динамическими элементами у нас являются ссылки. Причем, имеются также дополнительные ссылки **Сохранено в кэше** и **Похожие страницы**, которые присутствуют не во всех узлах.В общем, фрейм обещает быть интересным.

### 4.3.1. Описание страницы

Стартовую страницу описать не составляет труда, поэтому мы на ней сейчас останавливаться не будем, тем более, что основная работа должна развернуться вокруг списка ссылок, а там возможно понадобятся какие-то специфические настройки. Итак, введем ключевое слово **TEST** и начнем поиск. Высветится список ссылок. Попробуем его записать. И тут начинаются трудности. Если уровень распознавания таблиц с нулевой толщиной линий установлен в значение, отличное от максимального, то верхняя шапка окна просто не видна. Её как бы нет. Но вообще, желательно, чтоб она во фрейме присутствовала. Поэтому установим уровень распознавания таблиц с нулевой толщиной линий в максимальное значение 1,0. Опишем фрейм при таких настройках. Допустим страница имеет вид (по-крайней мере в данный момент, когда я описываю данный фрейм, страница имеет такой вид):

**Веб** Картинки Группы Каталог **Дополнительно »**

TEST Поиск Расширенный поиск Настройки

Искать: Интернет русскоязычные страницы страницы из Украины

**Веб** Результаты **1 - 10** из примерно **1 390 000 000** для **TEST**. (**0,17** секунд)

Совет: Ищите страницы только на **русском** языке . Вы можете задать язык поиска в разделе Настройки

## **Test** Central Home

Provides extranet privacy to clients making a range of tests and surveys available to their human...
www.**test**.com/ - 39k - Сохранено в кэше - Похожие страницы

## **Test** of English as a Foreign Language (TOEFL)

Information about the TOEFL tests and services are available online. Try the TOEFL practice questions.
www.ets.org/toefl/ - 23k - Сохранено в кэше - Похожие страницы

## STIFTUNG WARENTEST online - Themen - **test** - FINANZtest

Herausgeber der Zeitschrift "**test**". Viele nutzliche Informationen, News und Testergebnisse.
www.stiftung-waren**test**.de/online/ - Похожие страницы

## IQTest.com

Free **test** with information on intelligence testing and history thereof. Additional detailed reports...
www.iq**test**.com/ - 10k - Сохранено в кэше - Похожие страницы

## Психологические тесты on-line

Тест на наркотики (для родителей) · Частная служба психологической помощи · NetИнформБюро: доска объявлений, банк резюме · @**test**@ - ПСИХОЛОГИЧЕСКИЕ ТЕСТЫ **...**
new**test**s.kulichki.net/ - 18k - Сохранено в кэше - Похожие страницы

## **Test** Prep Review - Your online resource for **test** preparation.

**Test** Prep Review - Your Source for Free Practice Tests **...** Adequate preparation time has become increasingly important as **test** takers lives are increasingly **...**
www.**test**prepreview.com/ - 38k - Сохранено в кэше - Похожие страницы

## www.oekotest.de (ho)

Die Internetseiten von OKO-**TEST**, dem kritischen Verbrauchermagazin zum Richtig gut leben.
www.oeko**test**.de/ - Похожие страницы

## Bandwidth Speed **Test**

**Test** your Internet connection, add a **test** to your website, view **test** statistics.
www.bandwidthplace.com/speed**test**/ - 15k - 5 июнь 2006 - Сохранено в кэше - Похожие страницы

## ACT, Inc. : Educational/Career Planning and Workforce Development

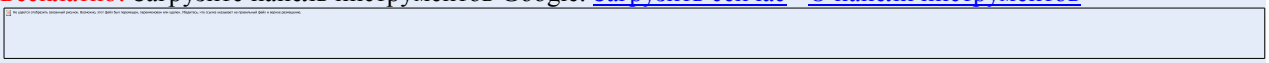
The organization responsible for the American College **Test**, required for admission to the public colleges...
www.act.org/ - 32k - 5 июнь 2006 - Сохранено в кэше - Похожие страницы

## Shields Up

GRC Internet Security Detection System scans on request the user's computer, especially the Windows...
https://grc.com/x/ne.dll?bh0bkyd2 - Похожие страницы

Страница результатов: 1 2 3 4 5 6 7 8 9 10 Следующая

Бесплатно! Загрузите панель инструментов Google. Загрузить сейчас - О панели инструментов



TEST Поиск

Поиск в найденном | Языковые инструменты | Как искать

---

Домашняя страница Google - Рекламные программы - Всё о Google



Фрейм изначально имеет вид:

**Code**

```
[+] window BrowserChild wGoogleResults
        [ ] tag "TEST -"
        [ ] parent Browser
        [ ]
        [+] HtmlTable HtmlTable1 // Здесь содержится форма ввода ключевых слов
и радио-кнопки поиска
                                                    // ( Интернет ,
русскоязычные страницы , страницы из Украины )
        ..........................................................................
..........................................................
        [ ]
        [ ]
        [+] HtmlTable HtmlTable2 // Пустая таблица. Её из фрейма можно будет
убрать
                [ ] tag "#2"
        ..........................................................................
..........................................................
        [ ]
        [ ]
        [+] HtmlTable HtmlTable3 // Содержит статистику поиска
                [ ] tag "#3"
                [-] HtmlColumn Веб
                        [ ] tag "Веб"
                [-] HtmlColumn Результаты110изпримерно
                        [ ] tag "Результаты 1 - 10 из примерно 1 390 000 000
для TEST. (0,17 секунд)"

        ..........................................................................
..........................................................
        [+] HtmlTable HtmlTable4 // Строчка с советами по поиску
```

```
                [ ] tag "#4"
        ..........................................................................
..........................................................................
        [ ]
        [+] HtmlLink TestCentralHome1
                [ ] tag "Test Central Home"
        [+] HtmlTable TestCentralHome2
                [ ] tag "Test Central Home"
                [+] HtmlColumn ProvidesExtranetPrivacyToC
                        [ ] tag "Provides extranet privacy to clients making a
range of tests and surveys available to their human...?www.test.com? - 39k -
Nio*"
                        [+] HtmlText ProvidesExtranetPrivacyToC
                                [ ] tag "Provides extranet privacy to clients
making a range of tests and surveys available to their human"
                        [+] HtmlText WwwTestCom39k
                                [ ] tag "www.test.com? - 39k -"
                        [+] HtmlText HtmlText3
                                [ ] tag "-"
                        [+] HtmlLink Сохранено в кэше
                                [ ] tag "Сохранено в кэше"
                        [+] HtmlLink Похожиестраницы
                                [ ] tag "Похожие страницы"
        [ ]
        [+] HtmlLink TestOfEnglishAsAForeignL1
                [ ] tag "Test of English as a Foreign Language (TOEFL)"
        [+] HtmlTable TestOfEnglishAsAForeignL2
                [ ] tag "Test of English as a Foreign Language (TOEFL)"
                [+] HtmlColumn InformationAboutTheTOEFLTe
                        [ ] tag "Information about the TOEFL tests and services
are available online. Try the TOEFL practice questions.?www.ets.org?toefl? -
23*"
                        [+] HtmlText InformationAboutTheTOEFLTe
                                [ ] tag "Information about the TOEFL tests and
services are available online. Try the TOEFL practice questions."
                        [+] HtmlText WwwEtsOrgToefl23k
                                [ ] tag "www.ets.org?toefl? - 23k -"
                        [+] HtmlText HtmlText3
                                [ ] tag "-"
                        [+] HtmlLink Сохраненовкэше
                                [ ] tag "Сохранено в кэше"
                        [+] HtmlLink Похожиестраницы
                                [ ] tag "Похожие страницы"
        [ ]
        ..........................................................................
..........................................................................
        [ ]
        [+] HtmlLink ShieldsUp
                [ ] tag "Shields Up"
        [+] HtmlTable ShieldsUp1
                [ ] tag "Shields Up[1]"
                [+] HtmlColumn GRCInternetSecurityDetectio
                        [ ] tag "GRC Internet Security Detection System scans
on request the user's computer, especially the
Windows...?https:??grc.com?x?ne.dl*"
                        [+] HtmlText GRCInternetSecurityDetectio
                                [ ] tag "GRC Internet Security Detection System
scans on request the user's computer, especially the Windows"
                        [+] HtmlText HttpsGrcComXNeDllBh0bk
                                [ ] tag "https:??grc.com?x?ne.dll?bh0bkyd2 -"
                        [+] HtmlLink Похожиестраницы
                                [ ] tag "Похожие страницы"
        [ ]
```

```
        [+] HtmlTable ACTIncEducationalCareer3  // Здесь содержится список
страниц результатов ( ссылки на номера страниц
                                                // и ссылки на предыдущую,
следующую страницы )
                [ ] tag "ACT, Inc. : Educational?Career Planning and Workforce
Development[2]"
                [+] HtmlColumn No?aieoa?acoeuoaoia
                        [ ] tag "No?aieoa ?acoeuoaoia:"
                [+] HtmlColumn HtmlColumn2
                        [ ] tag "#2"
                        [+] HtmlImage HttpWwwGoogleComUaIntl
                                [ ] tag "#1"
                [+] HtmlColumn N1 //
                        [ ] tag "1"
                        [+] HtmlImage HttpWwwGoogleComUaIntl
                                [ ] tag "#1"
                        [+] HtmlText N1
                                [ ] tag "1"
                [+] HtmlColumn N2 //
                        [ ] tag "2"
                        [+] HtmlLink N2
                                [ ] tag "2"
                        [+] HtmlImage HttpWwwGoogleComUaIntl
                                [ ] tag "#1"
        ..........................................................................
..........................................................

                [+] HtmlColumn N10
                        [ ] tag "10"
                        [+] HtmlLink N10
                                [ ] tag "10"
                        [+] HtmlImage HttpWwwGoogleComUaIntl
                                [ ] tag "#1"
                [+] HtmlColumn Neaao?uay // Содержит ссылку на следующую
страницу
                        [ ] tag "Neaao?uay"
                        [+] HtmlLink Neaao?uay
                                [ ] tag "Neaao?uay"
                        [+] HtmlImage HttpWwwGoogleComUaIntl
                                [ ] tag "#1"
        [+] HtmlTable ACTIncEducationalCareer4  // Ссылки "Загрузите панель
инструментов Google"
                [ ] tag "ACT, Inc. : Educational?Career Planning and Workforce
Development[3]"
        ..........................................................................
..........................................................
        [+] HtmlTable ShieldsUp2  // Нижняя форма для ввода ключевых слов и
ссылки Поиск в найденном | Языковые инструменты | Как искать
                [ ] tag "Shields Up[2]"
        ..........................................................................
..........................................................
        [+] HtmlTable ShieldsUp3 // Ссылки Домашняя страница Google -
Рекламные программы - Всё о Google
                [ ] tag "Shields Up[3]"
        ..........................................................................
..........................................................
        [+] HtmlText ©2006Google // Footer окна
                [ ] tag "©2006 Google"
```

### 4.3.2. Описание класса для работы со ссылками на найденные страницы

Как видно из листинга, фрейм еще нуждается в доработке. Более того, всвязи с максимальным уровнем распознавания таблиц с нулевой шириной границ многие

элементы фрейма находятся на достаточно глубоком уровне вложенности. Но до них мы доберемся потом. А сейчас обратим особое внимание на ту часть фрейма, которая в последнем листинге выделена красным. Это как раз и есть набор наших ссылок. Как видим, ссылки на найденные страницы находятся на 1-м уровне вложенности в иерархии и никаких других ссылок на данном уровне нет. Это очень даже неплохо, так как такое расположение позволяет нам обращаться к любой ссылке динамически, например:

**Code**

```
        [ ] Print( wGoogleResults.HtmlLink("#1").GetLocation( ) )
        [ ] wGoogleResults.HtmlLink("#1").Click( )
```

В данном примере мы обращаемся к первой ссылке.
Следующее наблюдение: текст ссылки (Caption) совпадает с тегом таблицы, содержащей ссылки "Сохранено в кэше" и "Похожие страницы", которые привязаны к данной найденной ссылке. То есть имеется способ добраться и до них ( причем этот способ вполне однозначный ). Таким образом ссылка на некоторую страницу, ссылки "Сохранено в кэше" и "Похожие страницы" формируют некоторый "узел", в который эти объекты можно сгруппировать. Попробуем подкорректировать один из таких узлов так, чтобы он минимально зависел от текста ссылки ( то есть обобщим структуру ). Возьмем для рассмотрения первый узел. Он выглядит так:

**Code**

```
        ..............................................................
..............................................................
        [+] HtmlLink TestCentralHome1
                [ ] tag "Test Central Home"
        [+] HtmlTable TestCentralHome2
                [ ] tag "Test Central Home"
                [+] HtmlColumn ProvidesExtranetPrivacyToC
                        [ ] tag "Provides extranet privacy to clients making a
range of tests and surveys available to their human...?www.test.com? - 39k -
Nio*"
                        [+] HtmlText ProvidesExtranetPrivacyToC
                                [ ] tag "Provides extranet privacy to clients
making a range of tests and surveys available to their human"
                        [+] HtmlText WwwTestCom39k
                                [ ] tag "www.test.com? - 39k -"
                        [+] HtmlText HtmlText3
                                [ ] tag "-"
                        [+] HtmlLink Сохраненовкэше
                                [ ] tag "Сохранено в кэше"
                        [+] HtmlLink Похожиестраницы
                                [ ] tag "Похожие страницы"
```

Ссылка на найденную страницу в данном случае первая по порядку и тег **"#1"** ей подойдет. Для сопряженной таблицы такой тег не подойдет, так как эта таблица не первая во фрейме. Можно, конечно, высчитать, сколько таблиц находится выше данной и скорректировать исходя из этого тег, но это не самый лучший вариант. Как уже указывалось выше, тег данной таблицы совпадает с **Caption** ссылки. А ссылку мы уже описали. Поэтому для таблицы тег можно реализовать как **"{this.HtmlLink("#1").GetFullCaption()}"**. Тэг будет определяться динамически, что нам, в принципе, и нужно. С внутренними объектами данной таблицы все проще: для колонки может быть использован тег **"#1"** ( колонка всего одна ), а теги ссылкок "Сохранено в кэше" и "Похожие страницы" неизменны. Также можно удалить ненужные элементы (HtmlText). Данный узел можно скорректировать так:

**Code**

```
        [+] HtmlLink TestCentralHome1
                [ ] tag "#1"
        [+] HtmlTable TestCentralHome2
                [ ] tag "{this.HtmlLink("#1").GetFullCaption()}"
                [ ]
                [+] HtmlColumn ProvidesExtranetPrivacyToC
                        [ ] tag "#1"
                        [+] HtmlLink Сохраненовкэше
                                [ ] tag "Сохранено в кэше"
                        [+] HtmlLink Похожиестраницы
                                [ ] tag "Похожие страницы"
```

Теперь обратим внимание на то, что для работы с данным узлом нам не нужны объекты HtmlTable и HtmlColumn, так как они только увеличивают вложенность вызова ссылок "Сохранено в кэше" и "Похожие страницы". Поэтому эти ссылки нужно подвинуть по иерархии на тот же уровень, что и главную ссылку узла. Также дадим объектам имена с латинскими буквами ( если в тегах кириллица еще не повредит, то идентификаторы все-таки лучше называть именами, состоящими из латинских букв), а также зададим им префиксы. Итак, окончательно узел преобразуется к виду:

**Code**

```
        [+] HtmlLink lnkLink
                [ ] tag "#1"
        [+] HtmlLink lnkCached
                [ ] tag
"[HtmlTable]{this.HtmlLink("#1").GetFullCaption()}/[HtmlColumn]#1/Сохранено в
кэше"
        [+] HtmlLink lnkRelativePages
                [ ] tag
"[HtmlTable]{this.HtmlLink("#1").GetFullCaption()}/[HtmlColumn]#1/Похожие
страницы"
```

Таким образом мы свели описание "узла" к описанию 3-х ссылок. Причем описание 1-го узла от остальных будет отличаться только индексом главной ссылки. Из этого следует, что мы выработали некоторый каркасс "узла", у которого есть только один параметр, определяющий, с каким же узлом нужно будет работать. Такая структура позволяет создать некоторый объект, который будет содержать интерфейс для обращения к тому или иному элементу того или иного "узла".

Создадим такой объект. Но для начала попытаемся описать, что же мы от него хотим. Итак, желательно, чтоб данный объект позволял работать как с отдельно взятым узлом ( выполнять операции перехода по ссылке, извлечение текста ссылки, переход на похожие страницы и т.д. ), так и со всеми узлами ( извлечь число узлов, извлечь ссылки для всех узлов и т.п. ). То есть данный объект должен работать со всем имеющимся списком узлов. Соответственно в окне **wGoogleResults** ( окно результатов поиска ) такой объект может быть представлен в единственном экземпляре. Данный объект реализуем в виде оконного класса. Декларация выглядит примерно так:

**Code**

```
[ ] winclass GoogleNodeList : AnyWin
```

Реально данный объект окна не представляет, но наследование от класса **AnyWin** позволит использовать целый ряд методов данного базового класса. Теперь опишем ключевые методы. Безусловно, это методы, которые возвращают сами объекты ссылок. Для начала опишем метод, который возвращает окно ссылки на найденную страницу. Данная ссылка может быть найдена по порядковому номеру либо по тексту ссылки. То есть параметром рассматриваемого метода может быть либо строка, либо число. В главе 2 при рассмотрении объектов списков, в частности при рассмотрении метода **Select** упоминался такой тип как **LISTITEM**, который позволяет хранить либо числовое либо

строковое значение. Данный тип подойдет и для рассматриваемого метода. Принцип действия нужного нам метода заключается в том, чтобы вернуть **this.HtmlColumn("#{iItem}")**, если параметром передано число, или вернуть **this.HtmlColumn( iItem )**, если параметром передана строка. Иначе остается сгенерировать исключение, так как ни один другой тип данных нам не подходит. Теперь рассмотрим параметр, если он передан в виде строки. Ссылки-то могут содержать различный текст, в котором могут оказаться специальные символы, например косая черта "/". Кстати на той странице результатов, что показана выше, такая ссылка как раз имеется. Если текст данной ссылки поставить в тег таким, каким он есть, то нужный объект не будет найдет, так как символ "/" в тегах является разделителем уровней окон. Поэтому подобные символы надо замаскировать. Заменим их всех символом "**?**" . Для этого опишем вспомагательную функцию вида:

**Code**

```
[+] private STRING Mask( STRING sInput )
        [+] LIST OF STRING lsSpec = {...}
                [ ] "/"
                [ ] """"
        [ ] STRING sValue
        [ ]
        [+] for each sValue in lsSpec
                [ ] sInput = StrTran( sInput , sValue , "?")
        [ ]
        [ ] return sInput
```

Пока что список маскируемых символов ограничивается только символами косой черты и кавычки. По мере необходимости этот список можно пополнять. Теперь у нас всё есть, чтобы описать метод класса **GoogleNodeList**, который возвращает ссылку на найденную страницу. А выглядит он так:

**Code**

```
[+] winclass GoogleNodeList : AnyWin
        ..............................................................
..............................................................
        [+] WINDOW GetLink( LISTITEM iItem )
                [+] switch TypeOf(iItem)
                        [+] case INTEGER
                                [ ] return this.HtmlLink("#{iItem}")
                        [+] case STRING
                                [ ] return this.HtmlLink( Mask( iItem ) )
                        [+] default
                                [ ] raise -1,"Incompatible parameter type"
```

Следует также отметить, что данный метод позволяет извлекать ссылки на страницы по тексту ссылки, используя специальные символы "*****" и "**?**". Дополнительные ссылки **Сохранено в кэше** и **Похожие страницы** ищутся исходя из текста ссылки на страницу. Больше ничего не требуется и методы реализуются в виде:

**Code**

```
[+] winclass GoogleNodeList : AnyWin
        ..............................................................
..............................................................
        [+] WINDOW GetCached( INTEGER iItem )
                [ ] return
this.HtmlLink("[HtmlTable]{this.GetLink(iItem).GetFullCaption()}/[HtmlColumn]
#1/Сохранено в кэше")
        [ ]
```

```
        [+] WINDOW GetRelativePagesLink(INTEGER iItem )
                [ ] return
this.HtmlLink("[HtmlTable]{this.GetLink(iItem).GetFullCaption()}/[HtmlColumn]
#1/Похожие страницы")
```

Следующей функциональностью, которую не помешало бы реализовать, является проверка на существование тех или иных ссылок определенного узла. Естественно, что прежде чем пытаться пойти по некоторой ссылке, нужно проверить, что такая ссылка существует. С главной ссылкой проблем не возникает, так как нам достаточно извлечь объект этой ссылки и применить метод **Exists**. Код выглядит так:

**Code**

```
[+] winclass GoogleNodeList : AnyWin
        ..............................................................
..............................................................
        [+] BOOLEAN LinkExists( LISTITEM liItem )
                [ ] return this.GetLink(liItem).Exists()
```

С остальными ссылками узла дело обстоит чуть сложнее. Все из-за того, что ссылки **Сохранено в кэше** и **Похожие страницы** используют текст главной ссылки, которой может и не оказаться, что повлечет генерацию исключения при попытке извлечь текст главной ссылки. Данное исключение надо перехватить, так как для данной функции это всего лишь результат, подтверждающий, что искомой ссылки нет. Проверки на существование ссылок **Сохранено в кэше** и **Похожие страницы** реализуются так:

**Code**

```
[+] winclass GoogleNodeList : AnyWin
        ..............................................................
..............................................................
        [+] BOOLEAN LinkCachedExists( LISTITEM liItem )
                [+] do
                        [ ] return this.GetCached(liItem).Exists()
                [+] except
                        [ ] return FALSE
        [+] BOOLEAN LinkRelativePagesExists( LISTITEM liItem )
                [+] do
                        [ ] return this.GetRelativePagesLink(liItem).Exists()
                [+] except
                        [ ] return FALSE
```

При желании, можно добавить в данную группу функций опциональный параметр, определяющий время ожидания появления нужной ссылки. Этот параметр просто подставляется в метод **Exists**.

Следующим этапом нужно создать методы, позволяющие извлекать количество ссылок на найденные страницы и текст этих ссылок. Реализуем их через **property** таким образом:

**Code**

```
[+] winclass GoogleNodeList : AnyWin
        ..............................................................
..............................................................
        [+] property iLinksCount
                [+] INTEGER Get( )
                        [ ] INTEGER iValue = 1
                        [ ]
                        [+] while( this.LinkExists( iValue ) )
                                [ ] iValue++
                        [ ] return iValue-1
        [ ]
        [+] property lsLinkNames
```

```
                [+] LIST OF STRING Get()
                        [ ] LIST OF STRING lsValue = {""}
                        [ ] INTEGER iValue = this.iLinksCount
                        [ ]
                        [+] while( iValue )
                                [ ] ListInsert( lsValue , 1 ,
this.GetLink(iValue--).GetFullCaption() )
                        [ ] ListDelete(lsValue,ListCount(lsValue))
                        [ ] return lsValue
```

И теперь осталась группа методов для выбора ссылок. Причем следует учитывать, что ссылку можно выбрать по номеру или по индексу, открыть ссылку можно в том же окне или в новом ( в контекстном меню выбрать пункт **Открыть в новом окне**). Эти соображения позволяют определить набор параметров. Первый параметр - величина типа **LISTITEM** ( для вызовов функций GetLink и др. ). Второй параметр - **BOOLEAN**, определяющий нужно ли открывать ссылку в новом окне. Данный параметр фактически влияет только на то, как будет произведен клик на ссылке. Если нужно открыть ссылку в том же окне, то клик будет сделан прямо на ссылке, в противном случае нужно вызвать **PopupSelect** метод, который сделает выбор из контекстного меню. Реализация метода имеет вид:

**Code**

```
[+] winclass GoogleNodeList : AnyWin
        ..............................................................
..........................................................
        [+] VOID Select( LISTITEM liItem , BOOLEAN bNewWindow NULL optional )
                [ ] RECT rc
                [ ] WINDOW wWin = this.GetLink( liItem )
                [+] if( IsNull( bNewWindow ) || !bNewWindow )
                        [ ] wWin.Click() // Открыть в том же окне
                [+] else
                        [ ] rc = wWin.GetRect()
                        [ ]
this.PopupSelect(rc.xPos+rc.xSize/2,rc.yPos+rc.ySize/2,"#2") // Открыть в
новом окне
```

Следует заметить, что ссылка извлекается сразу, чтобы потом не дублировать вызовы **GetLink**, тем более что данную функцию можно дополнить различными проверками, например на то, что извлеченная ссылка существует. Также обратим внимание на вызов метода **PopupSelect**. Данный метод сработает для окна, которое имеет визуальное представление. Но описываемому нами классу не соответствует никакого окна. Но данный класс теперь наследуется от класса **AnyWin** и экземпляр рассматриваемого класса своего тега не имеет, соответственно метод **PopupSelect** сработает для базового окна. Если бы данный класс не наследовался от **AnyWin**, то пришлось бы напрямую извлекать родительское окно.

Опишем аналогичные методы для выбора остальных ссылок узла. Реализуются они идентично:

**Code**

```
[+] winclass GoogleNodeList : AnyWin
        ..............................................................
..........................................................
        [+] VOID SelectCached( LISTITEM liItem , BOOLEAN bNewWindow NULL
optional )
                [ ] RECT rc
                [ ] WINDOW wWin = this.GetCached( liItem )
                [+] if( IsNull( bNewWindow ) || !bNewWindow )
                        [ ] wWin.Click()
                [+] else
```

```
                        [ ] rc = wWin.GetRect()
                        [ ] this.PopupSelect(rc.xPos + rc.xSize/2,rc.yPos +
rc.ySize/2,"#2")
        [+] VOID SelectRelative( LISTITEM liItem , BOOLEAN bNewWindow NULL
optional )
                [ ] RECT rc
                [ ] WINDOW wWin = this.GetRelativePagesLink(liItem)
                [+] if( IsNull( bNewWindow ) || !bNewWindow )
                        [ ] wWin.Click()
                [+] else
                        [ ] rc = wWin.GetRect()
                        [ ]
this.PopupSelect(rc.xPos+rc.xSize/2,rc.yPos+rc.ySize/2,"#2")
```

Как видно, последние 3 метода отличаются только строкой инициализации переменной **wWin**. Естественно, можно эти функции сгруппировать в одну и добавить еще один параметр - код ссылки, позволяющий определить на какую из ссылок узла нужно кликнуть. Это уже кому как удобней. Я предпочту оставить эти методы в таком виде, так как модификации и корректировки для этих методов могут различаться.
Таким образом мы описали класс для работы со списком ссылок на найденные страницы. Это одна из самых сложных частей, поэтому мы реализовали ее в первую очередь.

### 4.3.3. Вынесение элементов фрейма вверх по иерархии

Теперь приведем в порядок остальные части фрейма. Первым делом мы удалим из фрейма объекты, которые мы вряд ли будем использовать. В нашем случае это

**Code**

```
[+] window BrowserChild wGoogleResults
        [ ] tag "TEST -"
        [ ] parent Browser
        [ ]
        ..............................................................
..........................................................
        [ ]
        [+] HtmlTable HtmlTable2
                [ ] tag "#2"
```

Выделенный красным фрагмент не содержит в себе объекты, с которыми можно что-то сделать. Это обычная таблица с одной колонкой и больше ничего. Уберем эту таблицу из фрейма.
Также скорее всего **HtmlTable HtmlTable4** не понадобится, так как в этой таблице содержится какая-то вспомогательная информация. Если по тесткейсам ссылки в этой панели используются, то тогда эту таблицу можно оставить. Я эту таблицу удалю, так как я с ней работать не собираюсь, да и она не всегда появляется.
Следующим этапом будет формирование более-менее удобной иерархии объектов во фрейме. То есть нужные объекты мы вынесем повыше в иерархии. С объектами, находящимися выше списка ссылок на найденные страницы, проблем не возникает, эти объекты описываются однозначно. Проблема возникает с теми секциями, которые находятся ниже списка ссылок. Все эти секции помещены в таблицы. У этих таблиц тег может варьироваться. Это зависит от текста ссылок ( варьируется тег по Caption ) и от количества ссылок (варьируется тег по индексу). Поэтому теги секций, следующих за списком ссылок, придется привязывать к списку ссылок. Наиболее оптимальным вариантом будет привязка по индексу, а именно: извлекаем индекс последней таблицы списка ссылок на найденные страницы и прибавляем к нему смещение. Для примера рассмотрим секцию, содержащую ссылки **Загрузить сейчас** и **О панели инструментов.** Элементы данной секции помещены внутри таблицы, индекс которой на 2 больше, чем индекс последней таблицы списка ссылок ( между ними еще есть секция выбора страниц

результатов ). Соответственно, чтобы определить тег элементов данной секции, нужно получить индекс последней таблицы списка ссылок и прибавить к этой величине 2. Для упрощения получения этих величин добавим в класс **GoogleNodeList** метод:

**Code**

```
[+] winclass GoogleNodeList : AnyWin
        ..............................................................
..............................................................
        [+] INTEGER GetLastTableIndex()
                [ ] return
this.HtmlLink("[HtmlTable]{Mask(this.GetLink(this.iLinksCount).GetFullCaption
())}[1]").iIndex
```

Для упрощения обращения к данному методу в пределах окна **wGoogleResults** допишем свойство **iBaseIndex**:

**Code**

```
[+] window BrowserChild wGoogleResults
        ..............................................................
..............................................................
        [+] property iBaseIndex
                [+] INTEGER Get()
                        [ ] return this.Nodes.GetLastTableIndex()
```

Теперь таблица, содержащая ссылки **Загрузить сейчас** и **О панели инструментов** может быть описана так:

**Code**

```
[+] window BrowserChild wGoogleResults
        ..............................................................
..............................................................
        [+] HtmlTable Table
                [ ] tag "#{iBaseIndex+2}"
```

Это было показано описание таблицы, но нам для фрейма нужно описать ссылки, находящиеся внутри данной таблицы. Эти ссылки вынесем во фрейме на самый верхний уровень, после чего ссылки описываются в виде:

**Code**

```
[+] window BrowserChild wGoogleResults
        ..............................................................
..............................................................
        [+] HtmlLink lnkDownloadNow
                [ ] tag "[HtmlTable]#{iBaseIndex+2}/[HtmlColumn]#1/Загрузить
сейчас"
        [+] HtmlLink lnkAboutToolbar
                [ ] tag "[HtmlTable]#{iBaseIndex+2}/[HtmlColumn]#1/О панели
инструментов"
        ..............................................................
..............................................................
```

По аналогии можно описать и другие секции, но тут появляется еще одна проблема - вышерассмотренная секция появляется только на первой странице поиска. Если мы переключимся на другую страницу, то данная секция там не будет отображаться и соответственно теги сместятся, в частности теги таблиц. Для этих целей удобно завести функцию, которая возвращает один из возможных вариантов тегов окна. В одном из файлов, содержащих общую функциональность, поместим функцию вида:

**Code**

```
[+] STRING OneOfTheTags( WINDOW wWindow, LIST OF STRING lsTags )
        [ ] STRING sValue
        [ ]
        [+] for each sValue in lsTags
                [+] if( wWindow.AnyWin(sValue).Exists() )
                        [ ] return sValue
        [ ] return NULL
```

Параметр **wWindow** принимает родительское окно для элемента, для которого нужно подобрать тег. Параметр **lsTags** содержит список тегов, из которых нужно выбрать подходящий. Используя новую функцию, можно описать элементы секции поиска внизу страницы ( ссылки **Поиск в найденном , Языковые инструменты , Как искать**, а также текстовое поле для ввода и кнопка поиска ). У них смещение может варьироваться между значениями 2 и 3. Соответственно эта секция описывается в виде:

**Code**

```
[+] window BrowserChild wGoogleResults
        ..............................................................
..............................................................
        [+] HtmlLink lnkSearchInFound
                [ ] tag
OneOfTheTags(WindowParent(this),{"[HtmlTable]#{iBaseIndex+3}/[HtmlColumn]#1/[
HtmlLink]Iiene a
iaeaaiiii","[HtmlTable]#{iBaseIndex+2}/[HtmlColumn]#1/[HtmlLink]Iiene a
iaeaaiiii"})
        [+] HtmlLink lnkLangInstr
                [ ] tag
OneOfTheTags(WindowParent(this),{"[HtmlTable]#{iBaseIndex+3}/[HtmlColumn]#1/?
cueiaua eino?oiaiou","[HtmlTable]#{iBaseIndex+2}/[HtmlColumn]#1/?cueiaua
eino?oiaiou"})
        [+] HtmlLink lnkHowToSearch
                [ ] tag
OneOfTheTags(WindowParent(this),{"[HtmlTable]#{iBaseIndex+3}/[HtmlColumn]#1/E
ae eneaou","[HtmlTable]#{iBaseIndex+2}/[HtmlColumn]#1/Eae eneaou"})
        [+] HtmlTextField edtSearch2
                [ ] tag
OneOfTheTags(WindowParent(this),{"[HtmlTable]#{iBaseIndex+3}/[HtmlColumn]#1/[
HtmlTable]#1/[HtmlColumn]#1/#1","[HtmlTable]#{iBaseIndex+2}/[HtmlColumn]#1/[H
tmlTable]#1/[HtmlColumn]#1/#1"})
        [+] HtmlPushButton btnSearch2
                [ ] tag
OneOfTheTags(WindowParent(this),{"[HtmlTable]#{iBaseIndex+3}/[HtmlColumn]#1/[
HtmlTable]#1/[HtmlColumn]#1/#1","[HtmlTable]#{iBaseIndex+2}/[HtmlColumn]#1/[H
tmlTable]#1/[HtmlColumn]#1/#1"})
```

Аналогично придется описывать секцию, содержащую ссылки **Домашняя страница Google , Рекламные программы , Всё о Google**, где смещение варьируется между 3 и 4. Для веб-приложений такое варьирование тегов не редкость. Страница может содержать кучу промежуточных элементов, которые не всегда, может, и нужны для автоматизации ( например рекламные баннеры ), но фрейм из-за них серьезно сдвигается. Поэтому такая вариация здесь вполне уместна.

### 4.3.4. Таблица ссылок на страницы результатов

Единственной секцией, которую мы еще не рассмотрели, является секция ссылок на страницы результатов. Фактически это таблица, содержащая ссылки на предыдущую, следующую страницы, а также ссылки на страницы по номерам. Данную секцию удобно реализовать отдельным оконным классом, чем мы сейчас и займемся. Данный класс формируется на основе таблицы, поэтому данный класс объявим в виде:

**Code**

```
[ ] winclass GooglePageNumbers : HtmlTable
```

Рассмотрим детально структуру данной таблицы. Первая колонка не содержит вообще функциональных элементов. Вторая колонка содержит ссылку на предыдущую страницу. Если открыта первая страница и, соответственно, ссылки на предыдущую страницу нет, то вторая колонка будет пустой, то есть 2-я колонка резервирует место для ссылки на предыдущую страницу. Следующие колонки содержат ссылки на номера страниц, на которые можно в данный момент переключиться. Последняя колонка содержит ссылку на следущую страницу. Таким образом у нас есть 2 фиксированные ссылки, которые мы можем сразу отразить во фрейме таким образом:

**Code**

```
[+] winclass GooglePageNumbers : HtmlTable
        [+] HtmlLink lnkPrevious
                [ ] tag "[HtmlColumn]#2/#1"
        [+] HtmlLink lnkNext
                [ ] tag "[HtmlColumn]#{this.GetColumnCount()}/#1"
```

Все остальное будет реализовано через функциональность. Например, для данного класса полезно извлекать список имен ссылок на страницы, доступные в данный момент. Это осуществляется перебором колонок, начиная с 3-й и до предпоследней. В этом интервале и содержатся ссылки. Нужно пройтись по всем колонкам в данном интервале и извлечь текст ссылок, которые находятся в этих колонках (если ссылки есть). Функциональность, реализующая данную операцию полностью укладывается в формат **property**, поэтому реализуем её так:

**Code**

```
[+] winclass GooglePageNumbers : HtmlTable
        ..........................................................................
.........................................
        [ ]
        [+] property lsLinks
                [+] LIST OF STRING Get()
                        [ ] LIST OF STRING lsValue
                        [ ] INTEGER i
                        [ ]
                        [ ] INTEGER iCount = this.GetColumnCount()
                        [ ]
                        [+] for i = 3 to iCount - 1
                                [+]
if(this.HtmlColumn("#{i}").HtmlLink("#1").Exists())
                                        [ ] ListAppend( lsValue ,
this.HtmlColumn("#{i}").HtmlLink("#1").GetFullCaption() )
                        [ ] return lsValue
```

А теперь проведем небольшой подсчет. Сколько раз данный фрагмент обратится к **this**? Один раз мы к нему обращаемся при извлечении количества колонок. Предположим, что в секции номеров страниц 10 номеров. Соответственно, обращений к **this** будет 19 ( одна из колонок содержит обычный текст вместо ссылки - это текущая страница). Итого 20 обращений. Это было бы не принципиально, если бы не одно обстоятельство. Данная секция следует сразу же за списком ссылок, поэтому тег у данной таблицы варьируется и имеет вид **"[HtmlTable]#{iBaseIndex+1}"**. Соответственно, при вызове свойства **lsLinks** 20 раз будет пересчитываться количество ссылок на странице. Это уже сейчас занимает много времени, но если предположить, что мы сделаем настройки , при которых будет на странице отображено 100 найденных ссылок, то время выполнения свойства **lsLinks** будет уже непозволительно большим. Чтобы как-то оптимизировать данное свойство, нужно хотя бы уменьшить количество обращений к **this**. Для этого нужно воспользоваться

какими-то другими объектами. Например, мы можем извлечь индекс данной таблицы, а также извлечь её родительское окно и уже через него обращаться к элементам таблицы. Это потребует 2 обращения к **this** вместо 20, что уже ускоряет работу. Перепишем данное свойство так:

**Code**

```
[+] winclass GooglePageNumbers : HtmlTable
        ..........................................................................
.........................................
        [ ]
        [+] property lsLinks
                [+] LIST OF STRING Get()
                        [ ] LIST OF STRING lsValue
                        [ ] INTEGER i
                        [ ]
                        [ ] STRING sTag = "#{this.iIndex}"
                        [ ] WINDOW wWin = this.GetParent()
                        [ ] INTEGER iCount =
wWin.HtmlTable(sTag).GetColumnCount()
                        [ ]
                        [+] for i = 3 to iCount - 1
                                [+]
if(wWin.HtmlTable(sTag).HtmlColumn("#{i}").HtmlLink("#1").Exists())
                                        [ ] ListAppend( lsValue ,
wWin.HtmlTable(sTag).HtmlColumn("#{i}").HtmlLink("#1").GetFullCaption() )
                        [ ] return lsValue
```

Таким образом мы избавились от обращения к **this** внутри цикла, что делает затраты времени на выполнение операции зависящими только от количества ссылок. Подобная ситуация часто будет возникать там, где объекты определяются динамически, как и в нашем случае. Поэтому код желательно оптимизировать, иначе будет неоправданная потеря времени при выполнении скриптов.
По аналогии с предыдущим свойством реализуем свойство, возвращающее номер текущей страницы поиска.

**Code**

```
[+] winclass GooglePageNumbers : HtmlTable
        ..........................................................................
.........................................
        [+] property iCurPage
        [+] INTEGER Get()
                [ ] INTEGER i
                [ ]
                [ ] STRING sTag = "#{this.iIndex}"
                [ ] WINDOW wWin = this.GetParent()
                [ ] INTEGER iCount = this.GetColumnCount()
                [ ]
                [+] for i = 3 to iCount - 1
                        [+]
if(wWin.HtmlTable(sTag).HtmlColumn("#{i}").HtmlText("#1").Exists())
                                [ ] return
Val(wWin.HtmlTable(sTag).HtmlColumn("#{i}").HtmlText("#1").GetFullCaption())
                [ ]
                [ ] return 0
```

И осталось теперь реализовать функцию выбора нужной страницы. Выбрать можно 2-мя способами:

- Указать номер колонки таблицы по порядку - это быстрый способ, удобен в тех случаях, когда в свое время список страниц уже был извлечен или нужно

постоянно выбирать самую первую/последнюю страницу из доступных в данный момент.

- Указать имя страницы, какое указано в тексте ссылки, на которую нужно кликнуть. Фактически передаем имя ссылки на страницу результатов

В данной функции тоже надо минимизировать обращение к **this**. Код реализуется в виде:

**Code**

```
[+] winclass GooglePageNumbers : HtmlTable
        ..........................................................................
.....................................
        [+] BOOLEAN Select( LISTITEM sPage )
                [ ] INTEGER iPage
                [ ]
                [ ] STRING sTag = "#{this.iIndex}"
                [ ] WINDOW wWin = this.GetParent()
                [ ]
                [+] switch( TypeOf( sPage ) )
                        [+] case INTEGER
                                [ ] iPage = sPage + 2
                        [+] case STRING
                                [ ] iPage = ListFind(
wWin.GooglePageNumbers(sTag).lsLinks , sPage ) + 2
                [ ]
                [+] if( iPage <= 2 )
                        [ ] Print("* * * Page ""{sPage}"" wasn't found")
                        [ ] return FALSE
                [ ]
                [+] do
                        [ ]
wWin.HtmlTable(sTag).HtmlColumn("#{iPage}").HtmlLink("#1").Click()
                [+] except
                        [ ] ExceptPrint()
                        [ ] return FALSE
                [ ]
                [ ] return TRUE
```

Все описано, теперь можно сводить. Но не забываем того, что это веб-приложение и его страницы делаются по некоторому шаблону, который в SilkTest-е соответствует оконному классу. Давайте, пройдемся по различным группам ссылок. Вверху страницы результатов поиска есть ссылки **Картинки, Группы, Каталог, Дополнительно**. Если пойти по первым трем ссылкам, то откроются страницы, у которых есть общая со страницей ссылок на найденные страницы составляющая - форма ввода критерия поиска в самом верху страницы. Эта часть может быть вынесена в оконный класс, который имеет вид:

**Code**

```
[+] winclass GoogleResultsBase : BrowserChild
        [ ]
        [+] HtmlRadioList rlstSearch
                [ ] tag
"[HtmlTable]#1/[HtmlColumn]#3/[HtmlTable]#2/[HtmlColumn]#1/Eneaou:"
        [+] HtmlTextField edtSearch
                [ ] tag
"[HtmlTable]#1/[HtmlColumn]#3/[HtmlTable]#1/[HtmlColumn]#1/[HtmlTable]#2/[Htm
lColumn]#1/Aiiieieoaeuii »"
        [+] HtmlPushButton btnSearch
                [ ] tag
"[HtmlTable]#1/[HtmlColumn]#3/[HtmlTable]#1/[HtmlColumn]#1/[HtmlTable]#2/[Htm
lColumn]#1/Iiene"
```

```
        [ ]
        [+] HtmlLink lnkExtSearch
                [ ] tag
"[HtmlTable]#1/[HtmlColumn]#3/[HtmlTable]#1/[HtmlColumn]#1/[HtmlTable]#2/[Htm
lColumn]#2/?anoe?aiiue iiene"
        [+] HtmlLink lnkSettings
                [ ] tag
"[HtmlTable]#1/[HtmlColumn]#3/[HtmlTable]#1/[HtmlColumn]#1/[HtmlTable]#2/[Htm
lColumn]#2/Iano?ieee"
        [ ]
        [+] HtmlLink lnkWeb
                [ ] tag
"[HtmlTable]#1/[HtmlColumn]#3/[HtmlTable]#1/[HtmlColumn]#1/[HtmlTable]#1/[Htm
lColumn]#1/Aaa"
        [+] HtmlLink lnkPictures
                [ ] tag
"[HtmlTable]#1/[HtmlColumn]#3/[HtmlTable]#1/[HtmlColumn]#1/[HtmlTable]#1/[Htm
lColumn]#1/Ea?oeiee"
        [+] HtmlLink lnkGroups
                [ ] tag
"[HtmlTable]#1/[HtmlColumn]#3/[HtmlTable]#1/[HtmlColumn]#1/[HtmlTable]#1/[Htm
lColumn]#1/A?oiiu"
        [+] HtmlLink lnkCatalog
                [ ] tag
"[HtmlTable]#1/[HtmlColumn]#3/[HtmlTable]#1/[HtmlColumn]#1/[HtmlTable]#1/[Htm
lColumn]#1/Eaoaeia"
        [+] HtmlLink lnkMore
                [ ] tag
"[HtmlTable]#1/[HtmlColumn]#3/[HtmlTable]#1/[HtmlColumn]#1/[HtmlTable]#1/[Htm
lColumn]#1/Aiiieieoaeuii »"
```

Соответственно, окно результатов поиска объявляется в виде:

**Code**

```
        [+] window GoogleResultsBase wGoogleResults
                [ ] parent Browser
                [ ] tag "$http:??www.google.com.ua?search?svnum=*hl=*"
                .....................................................
```

Обратите внимание на тег данного окна. Данное окно распознается по Window ID. Это достаточно уникальный идентификатор и в веб-приложениях он генерируется либо по адресу страницы, либо по атрибуту **name** элемента управления. Так или иначе, в веб приложениях такой тег является и уникальным и удобным для понимания, поэтому здесь мы используем его.

Теперь мы можем описать страницы, открываемые по ссылкам **Картинки, Группы, Каталог**, но мы рассмотрение этих окон пропустим, так как их написание также займет много времени и ничего нового все равно освещено не будет.

Единственное, что осталось перед тем как реализовать некоторый сценарий, - это описать стартовую страницу. Содержимое данного окна не меняется, поэтому фрейм не представляет сложности и описывается в виде:

**Code**

```
[+] window BrowserChild wGoogleStart
        [ ] tag "Google"
        [ ] parent Browser
        [ ]
        [+] HtmlLink lnkPictures
                [ ] tag "[HtmlForm]#1/[HtmlTable]#1/[HtmlColumn]#1/Ea?oeiee"
        [+] HtmlLink lnkGroups
                [ ] tag "[HtmlForm]#1/[HtmlTable]#1/[HtmlColumn]#1/A?oiiu"
        [+] HtmlLink lnkCatalog
```

```
                [ ] tag "[HtmlForm]#1/[HtmlTable]#1/[HtmlColumn]#1/Eaoaeia"
        [+] HtmlLink lnkMore
                [ ] tag
"[HtmlForm]#1/[HtmlTable]#1/[HtmlColumn]#1/Aiiieieoaeuii »"
        [+] HtmlRadioList rlstSearch
                [ ] tag "[HtmlForm]#1/[HtmlTable]#2/[HtmlColumn]#1/Eneaou:"
        [ ]
        [+] HtmlTextField edtSearch
                [ ] tag "[HtmlForm]#1/[HtmlTable]#2/[HtmlColumn]#2/#1"
        [+] HtmlPushButton btnGoogleSearch
                [ ] tag "[HtmlForm]#1/[HtmlTable]#2/[HtmlColumn]#2/Iiene a
Google"
        [+] HtmlPushButton btnLucky
                [ ] tag "[HtmlForm]#1/[HtmlTable]#2/[HtmlColumn]#2/Iia
iiaac?o!"
        [ ]
        [+] HtmlLink lnkExtSearch
                [ ] tag "[HtmlForm]#1/[HtmlTable]#2/[HtmlColumn]#3/?anoe?aiiue
iiene"
        [+] HtmlLink lnkSettings
                [ ] tag "[HtmlForm]#1/[HtmlTable]#2/[HtmlColumn]#3/Iano?ieee"
        [+] HtmlLink lnkLangInstr
                [ ] tag "[HtmlForm]#1/[HtmlTable]#2/[HtmlColumn]#3/?cueiaua
eino?oiaiou"
        [ ]
        [+] HtmlLink lnkAdvPrograms
                [ ] tag "?aeeaiiua i?ia?aiiu"
        [+] HtmlLink lnkAllAboutGoogle
                [ ] tag "An? i Google"
        [+] HtmlLink lnkGoogleComInEnglish
                [ ] tag "Google.com in English"
        [+] HtmlLink lnkMakeHomePage
                [ ] tag "Naaeaeoa Google noa?oiaie no?aieoae!"
```

Все, основа фрейма написана. Данные окна и оконные классы поместим в файл **Google.inc**, а для реализации скрипта создадим файл **Google.t**. Можно переходить к реализации тестовых сценариев.

## 4.4. Практика написания скриптов

Теперь рассмотрим особенности написания скриптов для тестирования веб-приложений. В принципе, реализуются они так же как и любые другие скрипты. То есть принципиальных различий, естественно, нет. Но есть моменты на которые нужно обратить внимание.

Во-первых, веб-приложение - это приложение имеющее клиентскую и серверную часть ( как правило ), причем сервер зачастую находится даже не в соседней комнате, а, допустим, в соседней стране или на соседнем континенте. То есть обработка запроса будет идти достаточно долго. Как результат - каждое уважающее себя веб-приложение ТОРМОЗИТ, соответственно, нужно как-то синхронизировать действия скрипта и работы приложения. Как минимум нужно предусмотреть то, что окна приложения могут долго открываться, а также долго закрываться. Для этих целей разработаем функцию, которая ждет появления/исчезновения некоторого окна. В файле **Common.inc** допишем функцию вида:

**Code**

```
[ ]
//*********************************************************************
**********************
[+] //* Function: BOOLEAN SynchWindow( WINDOW wWin , BOOLEAN bAppear NULL
optional)
```

```
        [ ] //* Description: This function waits untill window given as first
parameter appears or
        [ ] //*              disappears depending on second parameter
        [ ] //* Arguments: WINDOW wWin - window to wait
        [ ] //*            BOOLEAN bAppear - (optional) flag which defines
whether window should appear/disappear
        [ ] //*                                  TRUE by default
        [ ] //* Return values: TRUE - window is in expected state
        [ ]
//*************************************************************************
**********************
[+] BOOLEAN SynchWindow( WINDOW wWin , BOOLEAN bAppear NULL optional)
        [ ] INTEGER iTimeout = Agent.GetOption (OPT_APPREADY_TIMEOUT)
        [ ] HTIMER hTimer
        [ ]
        [+] if( IsNull( bAppear ) )
                [ ] bAppear = TRUE
        [ ]
        [ ] hTimer = TimerCreate(hTimer)
        [ ] TimerStart(hTimer)
        [ ]
        [+] while( TimerValue(hTimer) < iTimeout )
                [+] if( wWin.bExists == bAppear )
                        [ ] return TRUE
        [ ]
        [ ] TimerStop(hTimer)
        [ ] TimerDestroy(hTimer)
        [ ]
        [ ] return FALSE
```

Здесь есть один маленький нюанс. Время ожидания появления/исчезновения окна определяется исходя из опции Агента **Application Ready Timeout**. Безусловно, можно завести глобальную переменную, хранящую время ожидания, но в данном примере обойдемся без неё.

Теперь пора подготовить функциональность, которая приводит приложение в начальное состояние. У веб-приложений одно главное окно **Browser**. С одной стороны это неудобно, так как может быть открыто много окон браузера и среди них сложно отыскать нужное окно. Но с другой стороны, если нам нужно закрыть все открытые на данный момент окна браузера, то наличие единственного объекта очень даже удобно. А в нашем случае второе будет весьма кстати. Для приведения приложения в начальное состояние нам нужно:

- Закрыть все окна браузера
- Запустить браузер
- Ввести адрес стартовой страницы и дождаться её появления

Реализуется это в **appstate**, который мы напишем в виде:

**Code**

```
[+] appstate apsGoogle() basedon none
        [+] do
                [+] while( Browser.Exists() )
                        [ ] Browser.SetActive()
                        [ ] Browser.TypeKeys("")
        [+] except
                [ ] ExceptPrint()
        [ ]
        [+] if( GetTestcaseState() == TCS_ENTERING )
                [ ] Browser.Invoke()
                [ ] SynchWindow( Browser )
                [ ] Browser.Navigate("www.google.com")
```

```
                [ ]
                [+] if( !SynchWindow( wGoogleStart ) )
                        [ ] Error("Failed to Navigate the Google")
```

Эту функцию мы поместим в файл **Google.inc**. Теперь рассмотрим код повнимательнее. Во-первых, браузер закрывается нажатием комбинации клавиш Alt-F4. Причиной этому служит универсальность данного метода. Окна веб-приложения могут не иметь сисемного меню и системных кнопок, что сводит на нет все остальные способы закрытия окна. А данная комбинация клавиш является стандартной. Во-вторых, во время закрытия окон происходит перехват исключений. В принципе, сильно плохо не будет, если мы закроем не все окна. Они будут мешать только тем, что будут забирать на себя определенную память в зависимости от содержимого. Закрытие всех окон браузера - это просто желательная операция, поэтому мы можем двигаться дальше независимо от того, закрылись ли все окна или нет. В-третьих, блок открытия окна приложения запускается только при старте тесткейса. Это вполне рационально, так как стартовать приложение нам нужно только перед запуском тесткейса. Иначе, при запуске более одного тесткейса подряд, приложение будет запускаться на выходе из одного тесткейса и при старте следующего, то есть 2 раза подряд, что занимает время.
Следующим моментом выделим использование окна **Browser**. Вызов **Browser.Invoke()** осуществляет поиск пути к исполняемому файлу программы-браузера, после чего запускает браузер. Это стандартный метод, который самостоятельно может определить исходя из настроек SilkTest-а какой браузер нужно искать, где его искать и т.д. То есть нам не нужно выдумывать это самим. Весьма показательно использование **Browser.Navigate("www.google.com")**. Фактически данные метод вводит в адресной строке определенные текст и жмет Enter. По большому счету, данное окно уже имеет в наличии интерфейс, позволяющий избежать прямого обращения к элементам главного окна ( адресная строка, меню, кнопки перехода). Обращение к основным элементам главного окна завуалировано функциями. Это позволяет не отвлекаться на разные мелочи.
А теперь опишем небольшой сценарий ( некоторый минимум ), который мы будем автоматизировать. Сценарий имеет вид:

| № шага | Описание | Ожидаемые результаты |
|---|---|---|
| 1 | В строке поиска введите ключевое слово TEST и нажмите кнопку Поиск | • Откроется первая страница результатов поиска<br>• Заголовок окна будет содержать введенное ключевое слово<br>• Ссылка на предыдущую страницу отсутствует<br>• Введенное ключевое слово содержится либо в названии ссылки либо в текстке под ссылкой |
| 2 | Откройте каждую из ссылок в новом окне. Нажмите Alt-F4 , чтобы закрыть окно | Выбранная ссылка откроется в новом окне. Окно закроется |
| 3 | Нажмите на ссылку **Следующая** | Откроется 2-я страница, появится ссылка на предыдущую страницу. Введенное ключевое слово содержится либо в названии ссылки либо в текстке под ссылкой |
| 4 | Нажмите на ссылку на 3-ю страницу | Откроется 3-я страница. Введенное ключевое слово содержится либо в названии ссылки либо в текстке под ссылкой |
| 5 | Нажмите на ссылку | Откроется 2-я страница |

| | **Предыдущая** | |
|---|---|---|
| 6 | Нажмите на ссылку **Домашняя страница Google** | Стартовая страница откроется |

Итак, начнем реализовывать первый шаг. Из действий нужно только ввести ключевое слово и нажать на кнопку поиска, после чего дождаться появления окна результатов поиска. Это делается так:

**Code**

```
[+] testcase GoogleLinkNavigation() appstate apsGoogle
        ................................................
        [ ] wGoogleStart.edtSearch.sValue = "TEST"
        [ ] wGoogleStart.btnGoogleSearch.DoClick()
        [+] if( !SynchWindow(wGoogleResults) )
                [ ] Error("No results avaliable")
```

Проверяем заголовок окна. Он должен содержать введенное ключевое слово.

**Code**

```
        [+] if( !MatchStr("TEST*","{Browser.sCaption}") )
                [ ] Error("Main window has incorrect caption. Expected:
""TEST*"" . "+
                          "Actual: ""{Browser.sCaption}"" ",FALSE)
```

Проверяем, что открылась первая страница. В классе **GooglePageNumbers**, соответствующем списку страниц результатов поиска, соответствующее свойство реализовано. Поэтому:

**Code**

```
        [+] if( wGoogleResults.tblPages.iCurPage != 1 )
                [ ] Error("It is not first page!!!. Actual page:
{wGoogleResults.tblPages.iCurPage} ")
```

Проверяем, что ссылки на предыдущую страницу нет:

**Code**

```
                [+] if( wGoogleResults.tblPages.lnkPrevious.Exists() )
                        [ ] Error("Link to the first page is avaliable",FALSE)
```

А теперь, достаточно сложная проверка. Нам нужно перебрать все ссылки и все тексты под ними и убедиться, что хотя бы что-то содержит ключевое слово. Ссылки извлечь можно, для этого и создавался класс **GoogleNodeList**, но функциональности по извлечению объектов текста под ссылкой нет. Её нужно дописать. Эти тексты находятся на том же уровне, что и ссылка на найденную страницу. Причем количество текстовых объектов варьируется. Это связано с тем, что ключевые слова, выделенные жирным шрифтом, воспринимаются как отдельные объекты. Итак, зайдем в файл **Google.inc** раскроем класс **GoogleNodeList** и допишем ему метод:

**Code**

```
[+] winclass GoogleNodeList : AnyWin
        ..........................................................
        [+] LIST OF WINDOW GetTexts(LISTITEM iItem)
                [ ] LIST OF WINDOW lwWins
                [ ] WINDOW wWin
                [ ] INTEGER iValue = 1
                [ ] wWin =
this.HtmlColumn("[HtmlTable]{Mask(this.GetLink(iItem).GetFullCaption())}[1]/[
HtmlColumn]#1").HtmlText("#{iValue}")
                [+] while( wWin.Exists() )
```

```
                        [ ] ListAppend(lwWins,wWin)
                        [ ] iValue++
                        [ ] wWin =
this.HtmlColumn("[HtmlTable]{Mask(this.GetLink(iItem).GetFullCaption())}[1]/[
HtmlColumn]#1").HtmlText("#{iValue}")
                [ ] return lwWins
```

Данный метод вернет список окон - объектов текста. Все, перейдем опять к написанию скрипта. Заметим, что проверка на наличие ключевых слов встречается в 3-х местах. То есть мы данный шаг можем реализовать в виде функции. Выше тесткейса впишем код:

**Code**

```
[+] private VOID SearchForKeyWord( STRING sKeyWord )
        [ ] LIST OF WINDOW lwWins
        [ ] LIST OF STRING lsLinks
        [ ] INTEGER iValue
        [ ] STRING sValue
        [ ]
        [ ] lsLinks = wGoogleResults.Nodes.lsLinkNames
        [ ] iValue = ListCount( lsLinks )
        [ ]
        [+] while( iValue )
                [ ] lwWins = wGoogleResults.Nodes.GetTexts(iValue)
                [ ] sValue = lwWins[1].GetFullCaption()
                [ ]
                [+] if( !MatchStr( "*{sKeyWord}*",sValue) &&
!MatchStr("*{sKeyWord}*",lsLinks[iValue]) )
                        [ ] Error("Neither link ""{lsLinks[iValue]}"" nor
descriptive text ""{sValue}"" contain ""{sKeyWord}"" text ",FALSE)
                [ ]
                [ ] iValue--
```

В этой функции мы добавили параметр - ключевое слово для поиска. Можно было, конечно, использовать статическое значение, но так лучше, так как мы можем потом параметризировать тесткейс.
Используем функцию и допишем данный шаг:

**Code**

```
        [ ] SearchForKeyWord( "TEST" )
```

Следующий шаг будет открывать все ссылки в новом окне. Класс **GoogleNodeList** уже снабжен методом **Select**, который позволяет выполнить данную операцию. Сложность возникает в том, чтобы проверить, что некоторая страница открылась. Дело в том, что мы не имеем представления о том, что за страница должна открыться вообще. Мы не знаем этих страниц, но как-то проверить это нужно. Но одно известно - при нажатии на ссылке страница будет загружаться. Мы можем дождаться, пока она загрузится, а потом закрыть ее. Если это была страница результатов поиска ( то есть в новом окне ссылка не открылась ), то тесткейс досрочно завершит работу, так как закрыто главное рабочее окно. Более того, после закрытия некоторой страницы нам нужно дождаться, пока страница результатов поиска вновь активируется. В этом шаге нам понадобятся переменные. Во-первых, нам нужно перебрать все ссылки, соответственно должен быть какой-то итератор. Добавим в начало тесткейса объявление **INTEGER iValue**. Далее, нам нужно ждать, когда ссылка будет открыта в новом окне и потом, когда страница результатов поиска активируется снова. В первом случае можно воспользоваться **Browser.WaitForReady** - стандартным методом, который ждет в течение некоторого промежутка времени пока страница загрузится. А вот во втором случае нам нужно самим фиксировать время по аналогии с функцией **SynchTime**. Разница лишь в том, что в данном случае нужно дождаться, пока свойство **bActive** не станет возвращать значение **TRUE**. Можно и это

реализовать функцией, но в нашем тесткейсе такой код нужен только в этом шаге, поэтому не будем на этом заостряться. Обратим внимание, что нам нужна переменная для таймера **HTIMER hTimer**. И еще одно. Время ожидания тоже надо как-то задавать. По аналогии с **SynchTime** создадим переменную **INTEGER iTimeout** и проинициализируем её некоторым фиксированным значением. Я задам его равным 30. У меня всё достаточно быстро работает, так что 30 секунд более чем достаточно. Итак, в блоке объявления переменных (в самом начале тесткейса) вписываем следующие строки:

**Code**

```
        [ ] HTIMER hTimer
        [ ] INTEGER iValue
        [ ] INTEGER iTimeout = 30
```

Всё, теперь можно реализовывать код шага:

**Code**

```
                [ ] iValue = wGoogleResults.Nodes.iLinksCount // Извлекаем
количество ссылок на странице
                [ ]
                [+] while( iValue )
                        [ ] wGoogleResults.Nodes.Select(iValue,TRUE) // В цикле
выбираем по индексу ссылку и открываем ее в новом окне
                        [+] do
                                [ ] Browser.WaitForReady() // Ждем, пока
страница загрузится
                        [+] except
                                [ ] Error("Page wasn't loaded during specified
timeout") // Если страница не загружается в течение промежутка времени,
                                                        // задаваемого опцией
Агента OPT_APPREADY_TIMEOUT, то сгенерируется исключение, означающее,
                                                        // что приложение не
готово и еще грузится
                        [ ]
                        [ ] Browser.TypeKeys("") // Закрываем страницу нажатием
Alt-F4
                        [ ]
                        [ ] hTimer = TimerCreate()
                        [ ] TimerStart(hTimer)
                        [ ]
                        [+] while( !wGoogleResults.bActive )  Ждем, пока
страница результатов поиска не активизируется вновь
                                [+] if( TimerValue(hTimer) > iTimeout )
                                        [ ] Error( "Search Results window
wasn't activated during specified timeout" )
                        [ ]
                        [ ] TimerStop(hTimer)
                        [ ] TimerDestroy(hTimer)
                        [ ]
                        [ ] iValue--
```

Далее идут действия попроще. Нужно переходить по ссылкам. Итак, кликаем на ссылку **Следующая** и проверяем, что следующая страница открылась. У объекта **wGoogleResults.tblPages**, соответствующего таблице со ссылками на страницы результатов поиска, есть свойство **iCurPage**. Для определения того, что переход на следующую страницу осуществился, поместим ожидание нужного значения в цикл с таймером по аналогии, как мы ждали активации страницы результатов поиска в предыдущем шаге. Делается это так:

**Code**

```
        [ ] wGoogleResults.tblPages.lnkNext.Click()
        [ ]
        [ ] hTimer = TimerCreate()
        [ ] TimerStart(hTimer)
        [ ]
        [+] while( wGoogleResults.tblPages.iCurPage != 2 )
                [+] if( TimerValue(hTimer) > iTimeout )
                        [ ] Error("Second page wasn't activated")
        [ ] TimerStop(hTimer)
        [ ] TimerDestroy(hTimer)
```

Проверка на существование ссылки на предыдущую страницу сводится к использованию метода **Exists**, а проверку на содержание ключевых слов в тексте ссылки или описания к ссылке мы еще в первом шаге вынесли в отдельную функцию, соответственно дописываем код данного шага:

**Code**

```
        [+] if( !wGoogleResults.tblPages.lnkPrevious.Exists() )
                [ ] Error("Link to Previous page doesn't exist",FALSE)
        [ ]
        [ ] SearchForKeyWord( "TEST" )
```

Следующий шаг реализуется аналогично ( только существование предыдущей ссылки проверять уже не надо ). Единственной разницей является то, что на нужную страницу результатов поиска надо перейти нажатием на номер желаемой страницы. А данная операция реализуется методами объекта **wGoogleResults.tblPages**. Код имеет вид:

**Code**

```
        [ ] wGoogleResults.tblPages.Select("3")
        [ ]
        [ ] hTimer = TimerCreate()
        [ ] TimerStart(hTimer)
        [ ]
        [+] while( wGoogleResults.tblPages.iCurPage != 3 )
                [+] if( TimerValue(hTimer) > iTimeout )
                        [ ] Error("Third page wasn't activated")
        [ ] TimerStop(hTimer)
        [ ] TimerDestroy(hTimer)
        [ ]
        [ ] SearchForKeyWord( "TEST" )
```

Следующий шаг еще более урезанный. Нужно только нажать на ссылку **Предыдущая** и проверить, что вторая страница открылась. Ничего больше проверять не надо. Код:

**Code**

```
                [ ] wGoogleResults.tblPages.lnkPrevious.Click()
                [ ]
                [ ] hTimer = TimerCreate()
                [ ] TimerStart(hTimer)
                [ ]
                [+] while( wGoogleResults.tblPages.iCurPage != 2 )
                        [+] if( TimerValue(hTimer) > iTimeout )
                                [ ] Error("Second page wasn't activated")
                [ ] TimerStop(hTimer)
                [ ] TimerDestroy(hTimer)
```

Остается последний шаг, который реализуется так:

**Code**

```
        [ ] wGoogleResults.lnkGoogleHomePage.Click()
        [ ]
```

```
        [+] if( !SynchWindow(wGoogleStart) )
                [ ] Error("Home Page wasn't activated")
```

И всё. Тесткейс написан. Вроде бы и тесткейс несложный, но обратите внимание, что много чего было готово изначально, когда мы описывали фрейм. Представьте, как бы вы проверяли текущую страницу, если бы во фрейме не было описано класса **GooglePageNumbers**. Это было бы намного тяжелее. А так оставалось только синхронизировать работу скрипта с работой приложения.

Таким образом мы на примере страницы Google рассмотрели работу с веб-приложением. Наиболее сложной частью таких приложений является фрейм, поскольку некоторая функциональная единица такого приложения представляет из себя, как-правило, составной объект содержимое которого меняется динамически ( как, например, список страниц в Google ). Более того, веб-приложения в большей мере способны варьировать содержимое своих окон, чем какое-либо другие приложение. Поэтому хорошо подготовленный фрейм - это залог хорошо написанных тесткейсов.

# 5. Распределенное, параллельное выполнение скриптов. Multitestcase

В силу различных обстоятельств может возникнуть необходимость выполнять некоторые действия параллельно, возможно даже на нескольких машинах или просто для запуска скриптов на удаленной машине. Организация SilkTest-а позволяет это реализовать. Дело в том, что у SilkTest-а есть разделения на т.н. **Host machine** (HM) и **target machine** (TM). С HM запускаются скрипты, активируются базовые настройки. Фактически это машина, на которой открыт SilkTest из которого производится запуск. На TM эти скрипты выполняются. На TM работает Агент, который принимает инструкции с HM и выполняет их на той машине, на которой Агент запущен. То есть, как минимум, на TM должен быть установлен Агент. Далее мы рассмотрим различные варианты использования такой конфигурации.

## 5.1. Запуск скрипта на удаленной машине

Это простейший вариант. Запуск на удаленной машине удобно производить в различных случаях, из которых можно выделить следующие:

- Наличие тестируемого продукта на некоторой удаленной машине при отсутствии SilkTest-а на ней (установлен только Агент).
- Тестируемый продукт работает лучше на удаленной машине или требуется протестировать именно на той машине. При этом на ней установлен только Агент или SilkTest работает нестабильно.
- На HM работает человек, которому помимо запусков скрипта нужно выполнять еще какие-то важные задачи. Поэтому его машина не должна быть полностью занятой во время тестирования.
- Просто шутки ради. Например сидит за некоторой машиной человек, который занимается всем чем угодно, но не работой и SilkTest с Агентом открыт только для отвода глаз (вроде как он с ними работает). В этом случае можно написать простенький скрипт, который, например, открывает Блокнот и пишет в нем что-то наподобие **"На работе бездельничать нехорошо"**. Подобный "холодный душ" подействует весьма отрезвляюще на данного работника, а вы получите массу удовольствия и воспитательный момент присутствует.

Это основные случаи, когда нужно запускать скрипты удаленно. Теперь рассмотрим условия, которые нужно обеспечить для такого запуска.

Во-первых, нужная машина должна находиться в сети, иначе вы никак не сможете передавать сигналы на нее.

Во-вторых, на удаленной машине должен быть открыт Агент. Именно он заставляет машину выполнять команды скрипта.

В-третьих, нужно настроить Агента так, чтобы он мог получать команды с другой машины. Для этого нужно выполнить следующие действия:

1. Активировать окно Агента
2. Кликнуть правой кнопкой мыши на заголовке и выбрать опцию **Network**.

3. В появившемся диалоге нужно выбрать используемый сетевой протокол ( в **Network:** поле ) и ввести номер порта - комбинацию из 4-х цифр. Сетевой протокол может быть **TCP/IP** или **NetBIOS**.
4. Нажмите OK.

Агент настроен.

В-четвертых, нужно установить соединение между HM и TM. Для этого на HM в SilkTest-е выбираем меню **Options > Runtime** В появившемся диалоге ввыбираем в поле **Network** используемый сетевой протокол. После этого, в поле **Agent Name** вводим имя машины, которая будет выполнять роль TM. Имя вводится в формате **<имя_машины>:<номер_порта>**. Номер порта должен совпадать с номером порта, который был установлен в Агенте на TM. После этого можно закрывать диалог нажатием OK.

Теперь, если вы запустите скрипт на HM, то выполнится он на той TM, на которую вы настроили. Если вы в скриптах испоьзуете расширения, то нужно позаботиться о том, чтобы на TM эти расширения тоже были подключены. Подключение расширений рассматриваются в главе **Использование расширений (ActiveX, Java, .NET, Explorer extensions)**. Если вы работаете с веб-приложениями, то нужно позаботиться еще и о том, чтобы настройки IE DOM расширений ( например уровень распознавания таблиц с нулевой границей, распознавание форм, текста и т.д.) имели необходимые значения.

Подключение HM к TM можно реализовать и програмно. Для этого нужно запустить Агент на TM, сделать ему необходимые настройки (см. выше). Допустим наша TM называется **TEST**, а порт Агента установлен в значение 1111. Тогда подключаться к TM можно следующим кодом:

**Code**

```
[ ] HMACHINE tm
[ ] HMACHINE host = GetMachine()
[ ]
[ ] tm = Connect("TEST:1111")
[ ] SetMachine(tm)
[ ]
[ ] // Enter your code
[ ]
[ ] SetMachine(host)
[ ] Disconnect(tm)
```

Рассмотрим этот пример детально. Во-первых, используются переменные типа **HMACHINE**. Величины данного типа являются дискрипторами. Эти дискрипторы используются в функциях для распределенной работы, а также для извлечения информации о той или иной машине. Диксриптор текущей машины можно получить с помощью функции **GetMachine()**. В данном примере мы задействовали 2 переменные-дискриптора. И одна из них, **host**, как раз содержит дискриптор главной машины. Для подсоединения к удаленной машине используется функция **Connect**, которая принимает параметром строку в формате **<имя_машины>:<номер_порта>**. Возвращаемое значение этой функции - дискриптор машины, к которой произвелось подключение. Для того, чтобы переключить выполнение скрипта на Агент на удаленной машине, используется функция **SetMachine**. Её вызов приводит к тому, что дальнейший код будет выполнен на машине, дискриптор которой передан параметром. В данном случае это дискриптор удаленной машины. То есть, дальнейший код, который должен находиться на месте закомментированного текста будет выполнен на машине, дискриптор которой хранится в

переменной **tm**. В данном случае это машина **TEST**. На ней начнет выполнять инструкции Агент, у которого номер порта 1111. Для разрыва соединения используется функция **Disconnect**, принимающая также дискриптор машины, от которой нужно отсоединиться. Вот таким образом можно запустить отдельный скрипт на удаленной машине.

## 5.2. Параллельное выполнение скриптов

Часто могут возникнуть случаи, когда некоторые фрагменты кода должны выполняться параллельно, отдельными потоками. Необходимость в этом может возникнуть в разных ситуациях, например, нужно выполнять какие-то действия до тех пор, пока не выполнятся какие-то внешние условия, которые нужно проверять отдельно. То есть имеется 2 потока, один из которых выполняет требуемые действия, а другой в это время проверяет на выполнение некоторого условия. И если это условие выполнено, то второй поток подает сигнал первому, что надо завершать выполнение.
Каждый такой поток в SilkTest-е может быть реализован в **spawn**-блоке. Этот блок содержит в себе код, который будет выполнен данным потоком. Выглядит генерация нескольких потоков так:

**Code**

```
[+] spawn
        [ ] // Actions for the first thread
[+] spawn
        [ ] // Actions for the second thread
[+] spawn
        [ ] // Actions for the third thread
```

В данном примере код находящийся внутри каждого из **spawn**-блоков будет выполнен одновременно. Если быть точным, то эти блоки начнут одновременно выполняться. Если же нужно, чтобы код, следующий за данными блоками выполнился только по завершении выполнения всех потоков, то для этих целей используется оператор **rendezvous.** Соответственно, в коде вида:

**Code**

```
[+] spawn
        [ ] // Actions for the first thread
[+] spawn
        [ ] // Actions for the second thread
[+] spawn
        [ ] // Actions for the third thread
[ ] rendezvous
[ ] // Other actions
```

участок кода, на месте которого находится комментарий // **Other actions**, выполнится только после завершения последнего из вышесгенерированных потоков.
Теперь допустим, что у нас есть несколько потоков, каждый из которых заключается в работе некоторой функции. То есть весь код некоторого потока заключается в вызове некоторой функции. В этом случае более удобным и компактным будет использование оператора **parallel**. Допустим, у нас есть функции **MyFunction()** и **AnotherFunction()**, которые нужно выполнять параллельно. Реализуется это так:

**Code**

```
[+] parallel
        [ ] MyFunction()
        [ ] AnotherFunction()
```

Этот код полностью идентичен следующему:

**Code**

```
[+] spawn
        [ ] MyFunction()
[+] spawn
        [ ] AnotherFunction()
```

```
        [ ] rendezvous
```

Как видно из 2-х последних примеров, использование **parallel** делает код более компактным.

А теперь представим ситуацию, когда есть несколько потоков и один из них является управляющим. Пока все потоки выполняют определенные действия, управляющий поток следит за системой и координирует действия других потоков. Соответственно, если данный поток вдруг должен быть остановлен, то другие потоки тоже должны остановить выполнение независимо от того, на какой стадии выполнения они находились. Исключение генерировать в данном случае нежелательно, так как это может иметь непредсказуемые последствия. В этом случае такой управляющий поток реализуется в **critical**-блоке. Пример реализации:

**Code**

```
        [+] spawn
                [ ] MyFunction()
        [+] critical
                [ ] AnotherFunction()
        [ ] rendezvous
```

Следует отметить, что в любой момент выполнения скрипта должно выполняться не более одного **critical**-блока.

Теперь следует обратить внимание на работу с переменными при параллельном выполнении потоков. В следующем примере

**Code**

```
        [ ] INTEGER iValue
        [ ]
        [+] spawn
                [ ] iValue = 5
        [+] spawn
                [ ] iValue = 3
        [ ] rendezvous
```

локальная переменная **iValue** будет иметь значение 5 в одном потоке и значение 3 в другом потоке. При этом она будет иметь разные значения одновременно. По-крайней мере это можно воспринимать именно так, поскольку изменение данной переменной в одном потоке никак не сказывается на значение этой переменной в другом потоке. Например, такой код:

**Code**

```
        [ ] INTEGER iValue=2
        [ ]
        [+] spawn
                [ ] iValue = 5
        [+] spawn
                [ ] iValue++
        [ ] Print( iValue )
        [ ] rendezvous
```

В файл результата все-равно введется значение 2, хотя казалось бы оно менялось в **spawn**-блоках. А что если нужно в каждом потоке обращаться к некоторой переменной, значение которой может быть доступно другому потоку? Или, что делать, если нужно внутри каждого потока занести некоторые результаты в переменную ( список, например ), чтобы потом можно было получить их значения вне потоков? То есть нужна переменная, значение которой было бы доступно после выполнения потоков. Обычная глобальная переменная в этом не поможет, так как может возникнуть ситуация, когда несколько потоков одновременно могут попытаться изменить значение этой переменной. Поэтому нужна переменная, которая бы требовала дополнительно доступ к своему содержимому. Для таких целей служит ключевое слово **share**. Оно применяется к внешним переменным. Особенностью переменных с таким модификатором является то, что к ним нужно

открывать доступ с помощью ключевого слова **access**. Ключевое слово **access** формирует блок, внутри которого возможен доступ к переменной. Пример использования:

**Code**

```
[ ] share INTEGER iValue=2
[ ]
[+] main()
        [ ]
        [ ]
        [+] spawn
                [+] access iValue
                        [ ] iValue = 5
        [+] spawn
                [+] access iValue
                        [ ] iValue++
        [ ] rendezvous
        [+] access iValue
                [ ] Print( iValue )
```

В результате работы данного скрипта в файл результатов запишется значение 6. То есть переменная **iValue** вначале была изменена первым потоком, а затем вторым. То есть каждый из потоков смог изменить значение внешней переменной. Обратите внимание, что если **access**-блок содержит в себе **spawn**-блоки, то внутри последних доступа к переменной уже не будет. Вот в этом примере:

**Code**

```
[ ] share INTEGER iValue=2
[ ]
[+] main()
        [+] access iValue
                [+] spawn
                        [ ] iValue = 5
                [+] spawn
                        [ ] iValue++
                [ ] rendezvous
                [ ] Print( iValue )
        [ ]
        [ ] return
```

переменная **iValue** после завершения работы потоков все равно останется равной 2.

## 5.3. Параллельное выполнение скриптов на нескольких машинах

А теперь рассмотрим комбинированный вариант, когда нужно запускать тесткейс на нескольких машинах, причем на каждой из машин он должен выполняться параллельно. В этом случае нужно, чтобы на всех машинах, на которых будет работать скрипт, был запущен и настроен Агент. Это естественно. Теперь разобъем условно весь скрипт на секции:

- Подключение
- Выполнение
- Отключение

Этап подключения как правило имеет вид:

**Code**

```
        [+] LIST OF STRING lsMachines = {...}
                [ ] // Enum machine names
        [ ] STRING sMachine
        ..............................
        [+] for each sMachine in lsMachines
```

```
                [+] do
                        [+] if( !IsConnected(sMachine) )
                                [ ] Connect( sMachine )
                [+] except
                        [ ] ExceptPrint()
                        [ ] // Add exception handling here
```

В данном примере мы для каждой машины из списка **lsMachines** проверяем наличие подключения ( функция **IsConnected**) и если его нет, то подключаемся. На этом этапе желательно перехватывать исключения, которые как правило сгенерируются в случае невозможности подключиться к той или иной машине. Такая ситуация может быть критичной, а может и нет. Например, если к машине не удалось установить подключение, то эту машину можно изъять из списка машин. Также на данном этапе можно установить подключение к некоторой главной машине. Это полезно в том случае, если скрипт ведет вывод промежуточных результатов или просто делает запись в некоторый файл. Чтобы не разбрасывать их по всем машинам, лучше эти файлы складывать в одном месте на одной машине. Поэтому нужно обеспечить доступ к главной машине отдельно.
Этап выполнения характеризуется наличием **spawn**-блоков. Если на всех машинах выполняется один и тот же сценарий (например для проведения стрессового тестирования) , то эти **spawn**-блоки можно реализовать в цикле. Более того, сам сценарий удобно реализовать отдельной функцией. Но это уже вопрос конкретной реализации.
Итак, в каждом из **spawn**-блоков в первую очередь нужно переключиться на нужную машину при помощи функции **SetMachine** и только после этого выполнять действия. Функция **SetMachine** может принимать не только величину типа **HMACHINE**, но и **STRING**. Допустим в переменной **sMachine** хранится имя машины, на которой будет запущен очередной поток. Тогда код имеет вид:

**Code**

```
        [+] spawn
                [+] do
                        [ ] SetMachine(sMachine)
                        [ ] // Add code here
                [+] except
                        [ ] ExceptPrint()
                        [ ] // Add exception handling here
```

Это примерный каркасс, которого имеет смысл придерживаться.
Этап отключения представляет собой цикл, в котором происходит отсоединение от удаленных Агентов. Этот этап имеет вид:

**Code**

```
        [+] for each sMachine in lsMachines
                [+] do
                        [+] if( IsConnected(sMachine) )
                                [ ] Disconnect( sMachine )
                [+] except
                        [ ] ExceptPrint()
                        [ ] // Add exception handling here
```

или еще проще

**Code**

```
        [ ] DisconnectAll()
```

Это были основные моменты. Теперь рассмотрим более специфические случаи. Допустим нам нужно реализовать сценарий, по которому на разных машинах выполняются разные действия, причем начальные состояния тоже различаются. Реализацию можно построить так:

1. Приведение системы в начальное состояние можно реализовать в **appstate**, который перебирает все машины и выполняет действия, необходимые на каждой из них. При этом подключение/отключение к машинам реализовать в этом **appstate**.
2. Параллельные действия, выполняемые на каждой машине реализуются в отдельном **spawn**-блоке.

Это вполне приемлемо и будет работать, но вот такое использование **appstate** несколько неудобно, так как список машин нужно резервировать как внешнюю переменную. Более того, если приведение в начальное состояние на каждой из машин соответствует некоторому **appstate** ( то есть на каждой машине нужно запустить определенный **appstate**), то имеет смысл поискать более удобный способ, заключающийся просто в передаче имен **appstate**. Для такого сложного взаимодействия между тесткейсом и **appstate** при распределенном выполнении скриптов существует специальный вид тесткейсов, т.н. **multitestcase**.
**Multitestcase** - (в дальнейшем "мультитесткейс") это модификатор функции, который определяет, что данная функция не возвращает значения и при этом принимает агрумент типа **LIST OF STRING**. По своему применению такие функции аналогичны **testcase**-функциям. Мультитесткейс тоже не может вызывать другой тесткейс и не может быть вызван тесткейсом, но может быть вызван обычной функцией. Но мультитесткейс отличается от тесткейса (помимо обязательного аргумента списка строк) тем, что к мультитесткейсу не привязывается **appstate**. Вполне логично, так как мультитесткейс соответствуем множеству в общем случае различных тесткейсов работающих распределенно и у каждого такого тесткейса свое начальное состояние. Но эта недостача компенсируется другими средствами. В частности, для привязки некоторого **appstate** к сценарию, который будет работать на конкретной машине, используется функция **SetUpMachine**. Эта функция подключается к некоторой ТМ и привязывает к ней **appstate**, который будет запущен на данной машине. Общий вид функции:

**SetUpMachine( sMachine , wMainWin , STRING sAppstateName )** , где

sMachine - имя машины, с которой ассоциируется данный **appstate**;
wMainWin - главное окно приложения на данной машине
sAppstateName - имя **appstate** Но эта функция только ассоциирует **appstate**, но не запускает. Чтобы запустить параллельно эти все **appstate** на соответствующих машинах, нужно использовать функцию **SetMultiAppStates()**. Таким образом, каркасс мультитесткейса выглядит следующим образом:

**Code**

```
[+] multitestcase MyMultiTestCase( LIST OF STRING lsMachines )
        [ ]
        [ ] SetupMachine(lsMachines[1],NULL,"apsFirstAppstate")
        [ ] SetupMachine(lsMachines[2],NULL,"apsSecondAppstate")
        ..................................................
        [ ]
        [ ] SetMultiAppStates()
        [ ]
        [+] spawn
                [ ] SetMachine(lsMachines[1])
                [ ] // Code to run on first machine
        [+] spawn
                [ ] SetMachine(lsMachines[2])
                [ ] // Code to run on second machine
        ..................................................
        [ ] rendezvous
        [ ]
        [ ] DisconnectAll()
```

Это общий вид. Естественно **apsFirstAppstate** и **apsSecondAppstate** - это существующие **appstate**. Если на всех машинах работает один сценарий и **appstate** у всех один и тот же, то тогда каркасс имеет вид:

**Code**

```
[+] multitestcase MyMultiTestCase( LIST OF STRING lsMachines )
        [ ] STRING sMachine
        [ ]
        [+] for each sMachine in lsMachines
                [ ] SetupMachine(sMachine,NULL,"apsAppstate")
        [ ]
        [ ] SetMultiAppStates()
        [ ]
        [+] for each sMachine in lsMachines
                [+] spawn
                        [ ] SetMachine(sMachine)
                        [ ] // Code to run
        [ ] rendezvous
        [ ]
        [ ] DisconnectAll()
```

Вариаций может быть много, но это основные случаи.

## 5.4. Выводы

Таким образом:

1. Запуск скриптов на удаленной машине может быть осуществлен путем настройки опций или програмным путем
2. Програмно распараллеливание потоков осуществляется при помощи ключевых слов **spawn**, **parallel**.
3. Ключевое слово **rendezvous** позволяет продолжить выполнение скрипта только после того, как завершатся все созданные потоки.
4. При распараллеливании потоков переменные при завершении выполнения потоков принимают значение до начала выполнения потока. То есть поток только во время своей работы может использовать измененные значения, но эти значения не сохраняются при завершении работы потока.
5. Ключевое слово **share** применяется для внешних переменных и позволяет им открывать доступ к своему значению (в том числе и для изменения) внутри потоков. Ключевое слово **access** непосредственно открывает доступ к значению переменной.
6. **multitestcase** - разновидность тесткейса, предоставляющая интерфейс для параллельного запуска скрипта на нескольких машинах.

# 6. Использование тестплана (Testplan)

**Тестплан (Testplan)** – это простое и удобное средство организации запусков ваших скриптов. С помощью тестпланов можно быстро выделить все либо несколько тесткейсов, которые необходимо запустить. Ниже показан пример тестплана.

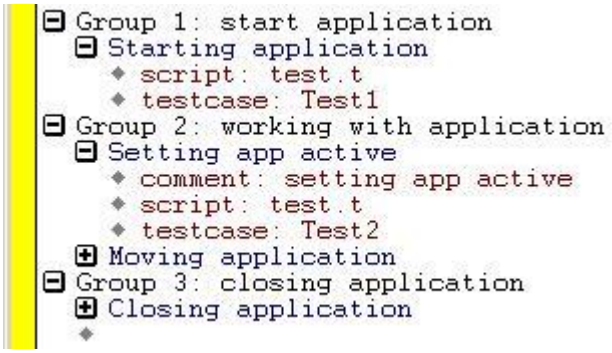

## Структура и типы тестпланов

В общем случае для добавления нового тесткейса в тестплан необходимо внести описание тесткейса (произвольная строка), файл, в котором находится тесткейс и, собственно, имя тесткейса. На приведенном выше скриншоте первые три строки как раз описывают один тесткейс. Его имя (*Starting application*), имя файла (*test.t*) и имя тесткейса (*Test1*).

Другой способ описания тесткейса показан на скриншоте ниже. Он начинает с символа "#" и не требует отдельного строкового описания. В качестве описания в этом случае выступает имя файла.

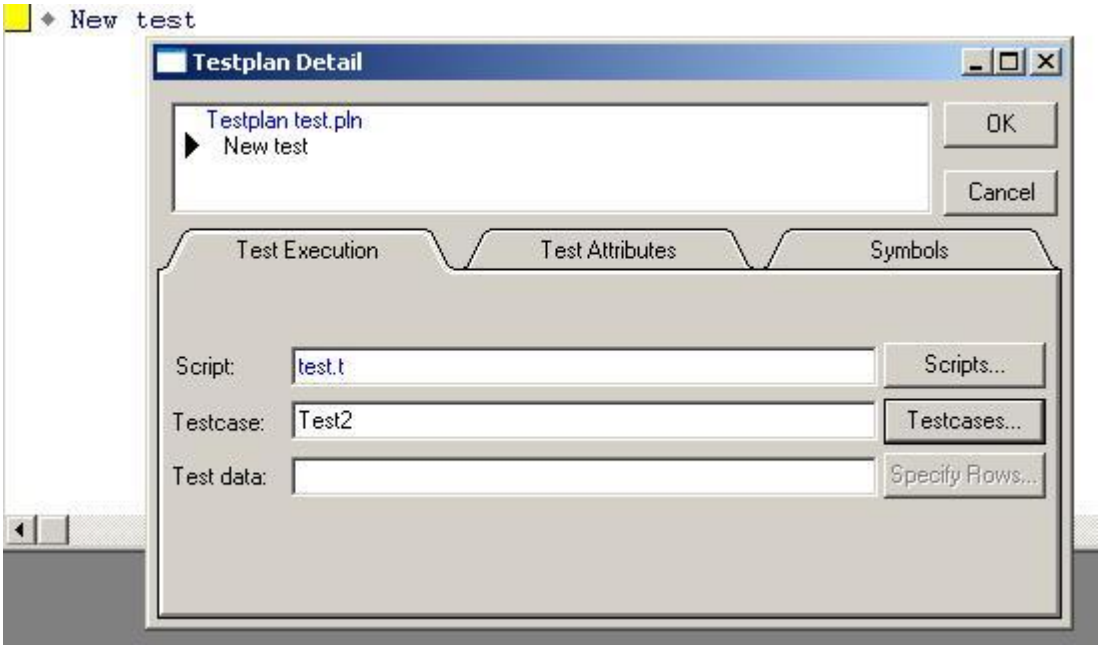
Как видно из скриншота, к тесткейсам в тестплане можно добавлять комментарии, которые затем будут отображаться в файле результатов сразу после текста лога скрипта. Комментарии, начинающиеся с двух слешей (//, отображаются зеленым цветом) в результаты не попадают и служат только для различных описаний в самом тестплане.

Тестпланы могут быть двух типов: простой и структурированный. В простом тестплане просто перечисляются тесткейсы в таком виде, как они описаны выше: имя, файл, тесткейс. В структурированном (или иерархическом) тестплане скрипты группируются по какому-либо признаку. Например, ниже показан пример тесткейсов, сгруппированных в три группы.




В случае использования иерархического тестплана в результатах также будет использоваться иерархическая структура. Добавлять данные в тестплан можно двумя способами: используя диалоговое окно *Testplat Detail* либо вручную.

Для добавления тесткейса с помощью диалогового окна необходимо вписать имя нового теста, выбрать пункт меню *Testplan - Detail* и в появившемся диалоговом окне выбрать имя файла и подходящий тесткейс и нажать ОК.




## Мастер план и вложенный тестплан

Тестпланы можно делать вложенными. **Вложенный тестплан (или субплан)** – это, фактически, отдельный файл тестплана, добавленный специальным образом в другой тестплан (называемый мастер планом). На первом скриншоте к данной главе показан пример вложенного тестплана *subpln.pln*.

Для того, чтобы добавить субплан в имеющийся мастер план, необходимо сначала создать отдельный файл тестплана (в нашем случае он называется subpln.pln), после чего установить курсор в то место тестплана, где по логике должен располагаться субплан, написать *"include: subpln.pln"* и нажать комбинацию клавиш Alt+F9. Если все сделано правильно, вложенный тестплан примет такой вид, какой показан на первом рисунке (выделится красным цветом и в конце пометится строкой ).

Обратите внимание на пустую строку перед строкой *include:* если ее не добавить, то тесткейс, находящийся непосредственно перед *include* блоком, выполняться не будет (очевидно, это бага в SilkTest).

Редактировать субпланы из мастер плана не допускается, для этого необходимо отдельно открыть субплан и внести в него изменения. После чего необходимо либо переоткрыть мастер план, либо выбрать пункт *Include - Acquire lock.*

Пункт меню *Acquire lock* позволяет также вносить изменения в субплан непосредственно из мастер плана. При этом субплан становится недоступным для остальных пользователей. После внесения изменений в субплан необходимо выбрать пункт меню *Release lock*, чтобы "освободить" вложенный план и дать возможность другим пользователям редактировать его. Доступен вложенный план для редактирования или нет, можно узнать по левой вертикальной черте в окне редактора: если тестплан доступен только для чтения – вертикальная черта будет серого цвета (как показано на первом скриншоте), если тестплан доступен для редактирования – черта будет желтого цвета. То же самое справедливо для всего тестплана (желтый цвет показывает доступность файла для редактирования).

## Выбор и запуск нескольких тесткейсов

Одно из главных назначений тестплана – запуск нескольких выбранных тесткейсов. Ниже показана панель инструментов SilkTest, предназначенная для выбора и запуска нескольких тесткейсов.

Описание кнопок панели инструментов (слева направо):

- **Goto the source...** – позволяет перейти к тесткейсу, на котором в данный момент находится курсор редактора
- **Modify detailed information...** – открывает диалоговое окно *Testplan Detail*, в котором можно изменить параметры текущего тесткейса
- **Add the current line...** – позволяет отметить текущий тесткейс для запуска
- **Remove the current line...** – удаляет текущий тесткейс из списка помеченных
- **Unmark the entire testplan** – снимает выделение для всех тесткейсов в тестплане
- **Find the next marked range** – переходит к следующей отмеченной группе тесткейсов в текущем тестплане
- **Compile...** – компилирование текущего тестплана
- **Find the next error** – перейти к следующей ошибке (кнопка доступна только когда открыт файл результатов)
- **Run the script** – позволяет запустить все тесткейсы подряд независимо от того, отмечены они для запуска или нет
- **Run...testcase** – запускает на выполнение или отладку отдельный тесткейс
- **Run the marked tests** – запускает только те тесты, которые отмечены в тестплане

На скриншоте ниже приведен пример выделенных тесткейсов (выделены тесткейсы Test2 и Test4):

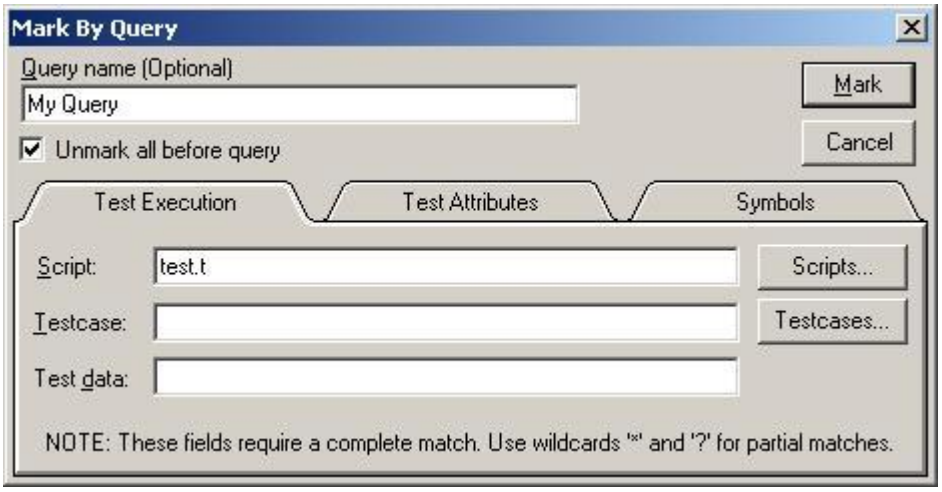
```
script: test.t
Starting application
  script: test.t
  testcase: Test1

include: subpln.pln
  Setting app active
    comment: setting app active
    script: test.t
    testcase: Test2
  Moving application
    script: test.t
    testcase: Test3

<eof>

#script: test.t // Another way for description
  comment: this testcase must be run the last
  testcase: Test4
Fifth testcase (manual)
  testcase: manual
```

Другой способ выделения нескольких тесткейсов – выделение с помощью запроса (**Query**). С помощью этого инструмента можно выделить, например, все тесткейсы из одного конкретного файла (если они разбросаны по тестплану и искать и выделять каждый из затруднительно), или выделить все тесткейсы по определенной маске (имени или части имени). Для этого необходимо выбрать пункт меню *Testplan – Mark by Query* и в появившемся диалоговом окне ввести параметры для выделения

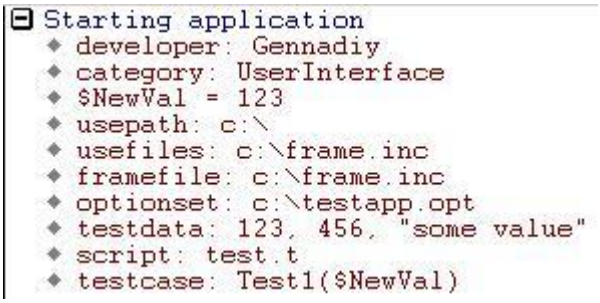



Если в поле *Query Name* ввести имя запроса, то оно сохранится и его можно будет использовать повторно, воспользовавшись пунктом меню *Testplan - Mark by Named Query*. Там же можно редактировать, удалять и объединять различные существующие запросы.

## Дополнительные возможности тестпланов

Из тестплана можно передавать в тесткейсы различные параметры, таким образом варьируя используемые в тесткейсах данные. Кроме того, в тестплане можно указывать различные тестовые атрибуты (категория теста, разработчик и т.п.), подключать файлы опций и т.п.

На скриншоте ниже мы постарались показать все возможные параметры, которые можно указывать в тестплане.




В этом примере указаны категория теста (*UserInterface*) и имя разработчика теста (*Gennadiy*), введена переменная *NewVal* (в терминах SilkTest такие переменные называются *символами*; обратите внимание на символ $ перед именем переменной). Эти параметры можно задать либо вручную, либо в диалоговом окне *Testplan Detail* (в случае добавления символов через диалоговое окно использовать символ $ не нужно), а отредактировать атрибуты можно, выбрав пункт меню *Testplan - Define Attributes*. Также здесь задаются путь для поиска подключаемых файлов (**usepath**), дополнительный подключаемый файл (**framefile, usefiles**) и передаются дополнительные тестовые данные (**testdata**).

Обратите внимание, что если вы передаете аргументы в тесткейс, сам тесткейс должен быть написан таким образом, чтобы принимать аргументы, так как нельзя передавать аргументы в функцию, которая не принимает никаких аргументов. Избежать этого можно двумя способами:

- используя необязательные аргументы с использованием ключевого слова **optional**
- используя произвольное количество аргументов (*varargs of ...*)

# 7. Обработка результатов (Results)

После запуска тесткейса, тестплана, или набора тесткейсов, SilkTest выводит окно с результатами отработанных тесткейсов. В нем отображается статистика по пройденным тесткейсам (общее количество, количество и процент тесткейсов, прошедших без ошибок - **Passed**, количество и процент прошедших с ошибками - **Failed**) и время, затраченное на прохождение всех тесткейсов. Также в этом окне отображается информация, выводимая в процессе работы скрипта функциями **Print**, **ListPrint**, **LogError**, **LogWarning** и другими. Для просмотра информации, выводимой в процессе работы тесткейса, необходимо нажать на значок [+] слева от названия тесткейса.

В файле результатов (с расширением ***.res** и таким же именем, как и название самого файла скрипта) хранятся результаты последних нескольких запусков. По умолчанию количество хранящихся результатов равно пяти, однако это количество можно изменить, зайдя в пункт меню *Options - Runtime* и установив в разделе **Results** в поле **History size** необходимое значение (от 0 до 32767). Кроме того, в этом же окне настроек в разделах **Results** и **Debug** можно изменить следующие опции:

- **Write to disk after each line** - если включено, то SilkTest будет записывать результаты в файл каждый раз, когда генерируется новая строка результатов. Это необходимо для того, чтобы файл результатов содержал все сгенерированные данные в случае аварийного заершения работы самого SilkTest-а или системы (например при отключении питания). Недостатком включения этой опции является то, что из-за постоянной работы с файлом результатов работа скрипта замедляется
- **Log elapsed time, thread, and machine for each output line** - если включено, то для каждой строки результатов будет выводиться информация о времени, прошедшем с начала запуска, имя текущего агента и номер параллельного процесса (подробнее о распараллеливании процессов читайте в главе 5. Распределенное, параллельное выполнение скриптов. Multitestcase). С помощью опций, находящихся в меню *Results - View Options* можно регулировать, какие именно из этих опций будут отображаться, а также отсортировать строки результата по одному из этих критериев
- **Find Error stops at warning** - если включено, то при переходе к следующей ошибке (меню *Edit - Find Error* или клавиша **F4**) будут осуществляться переходы как к ошибкам **(Error)**, так и к предупреждениям **(Warning)**. Если выключено, то переходы будут осуществляться только к ошибкам, игнорируя предупреждения
- **Show overall summary** - регулирует показывать или нет общую статистику по всем пройденным тесткейсам
- **Directory/File** - здесь можно указать папку, в которую SilkTest должен помещать сгенерированные файлы результатов, либо само имя файла результатов (по умолчанию файл результатов будет называться так же, как и файл скрипта)
- **Print agent calls** - если включено, то в файл результатов будут писаться все вызовы методов (их имена и передаваемы им параметры), которые происходят во время работы скрипта
- **Print tags with agent calls** - если включено, то при вызове методов будут также выводиться и теги используемых окон

Последние две опции полезны при отладке, так как в этом случае четко выводятся все теги и вызываемые методы (подробнее об отладке читайте в разделе 2.5 Отладка скрипта (debug)). Из-за большего количества выводимой информации при их включении размер файла результатов существенно увеличивается.

Еще некоторые пункты меню *Results*:

- **Select** - позволяет выбрать один из сохраненных ранее результатов (при открытии файла результатов по умолчанию открывается последний результат)
- **Merge** - позволяет добавить к текущему выбранному результату один из прошлых результатов
- **Delete** - удаляет один из имеющихся результатов
- **Extract** - позволяет извлечь результаты и поместить их в новый файл в окне тесткейса (пункт **Window**), который затем можно сохранить как текстовый файл, либо поместить результат сразу в файл (пункт **File**), либо вывести сразу на печать (пункт **Printer**)
- **Export** - позволяет конвертировать информацию о результатах (имя тесткейса, количество ошибок и их текст, количество предупреждений и т.п.) в текстовый файл с разделителями. Это может быть полезно для последующей перегонки результатов в базу данных или файл электронной таблицы
- **Send to Issue Manager** - позволяет переслать результаты непосредственно в **SilkCentral Issue Manager** (систему управления дефектами от **Segue**)
- **Convert to Plan** - конвертирует текущий файл результатов в файл тестплана. Данная опция доступна лишь в том случае, если файл результатов был сгенерирован при запуске отдельного тесткейса или скрипта (но не другого тестплана)
- **Show Summary/Hide Summary** - скрыть/показать суммарную информацию по каждому тесткейсу
- **Goto Source** - позволяет перейти к скрипту, откуда был вызван тесткейс. Если курсор находится в файле результатов в конкретном тесткейсе, то после открытия файла скрипта курсор будет находится в начале этого тесткейса. Если курсор в файле результатов находится на сообщении об ошибке, то при открытии файла скрипта курсор будет находиться в том месте, откуда было вызвано сообщение об ошибке
- **Mark Failures in Plan** - выделяет в тестплане только те тесткейсы, которые прошли с ошибками, после чего можно их перезапустить отдельно от остальных

Ниже приведен список функций для работы с результатами:

1. **Print(vaExprs)** - выводит в файл результата одну или несколько переменных любого типа, разделенных пробелами. Например:

**Code**

```
Print ("Window opened", 123, {"one", "two", "three"}, wTestApp.Exists())
```

2. В этом примере в файл результата выводятся: строка "Window opened", число 123, список строк {"one", "two", "three"} и результат, возвращаемый методом **Exists** окна **wTestApp** (TRUE или FALSE)
3. **ListPrint(lList)** - выводит в файл результата список, каждый элемент которого будет расположен в новой строке (в отличие от функции **Print**, которая выводит все элементы списка в одной строке, разделенные запятыми)
4. **Printf(sFormat, aArgs, ...)** - выводит форматированную строку **sFormat**. **aArgs** - любое количество аргументов. Сходна с функцией языка C **printf**. Здесь мы не будем подробно рассматривать эту функцию, приведем лишь небольшой пример:

**Code**

```
Printf ("Current time is %s, here are 5 spaces: '%5s', random integer value:
%d", TimeStr(), "", RandInt(1, 32000))
```

5. В результате будет выведена следующая строка:

**Code**

```
Current time is 22:04:18, here are 5 spaces: '     ', random integer value:
2178
```

6. Мы предоставим читателю самому разобраться в особенностях работы этой функции, воспользовавшись встроенной справкой.
7. **LogWarning (sWarning)** - выводит строку предупреждения (по умолчанию фиолетового цвета). Обычно используется для предупреждения пользователя о какой-либо некритичной ошибке
8. **LogError (sMsg)** - выводит сообщение об ошибке (по умолчанию красного цвета). Здесь необходимо отметить одну недокументированную особенность функции **LogError**, которая может оказаться очень полезной. Хотя в справке сказано, что эта функция принимает один параметр, на самом деле ей можно передать еще один строковый параметр. В случае, когда будет вызвана функция **LogError** с заданным вторым параметром, в файле результата возле собственно сообщения об ошибке (параметр **sMsg**) будет расположен красный квадратик. Если щелкнуть по этому квадратику, то SilkTest попытается запустить программу, имя которой должно быть передано в том самом втором недокументированном параметре. Например, после того, как сработает следующая функция

**Code**

```
LogError ("Here is calculator for you!", "calc.exe")
```

9. В файле результата будет выведена строка "Here is calculator for you!", слева от которой будет расположен квадратик. Если по нему щелкнуть, то откроется обычный калькулятор из стандартной поставки Windows (**calc.exe** - это имя исполняемого файла для калькулятора). Естественно, что второй параметр может быть любой программой, которую вам нужно запустить в данном случае. Достаточно простым и в то же время полезным действием в случае возникновения ошибки было бы делать скриншот экрана для того, чтобы во время разбора причины ошибки можно было бы увидеть, что именно отображалось на экране в момент ее возникновения. Захват изображения любого объекта осуществляется с помощью метода **CaptureBitmap()**. Для захвата изображения всего экрана используется объект **Desktop**. Итак, функция, которая захватывает изображение экрана и сохраняет его в **bmp-файл** с таким же именем, как и имя текущего тесткейса, будет выглядеть так:

**Code**

```
[-] STRING CreateScreenShot ()
        [ ] STRING sBitmap
        [ ]
        [-] withoptions
                [ ] BindAgentOption (OPT_BITMAP_MATCH_COUNT, 1)
                [ ] sBitmap = "{GetProgramDir ()}\{GetTestCaseName
()}_{FormatDateTime (GetDateTime (), "_ddmmyyyy_hhnnss")}.bmp"
                [ ] Desktop.CaptureBitmap (sBitmap)
        [ ] return sBitmap
```

10. Эта функция создает файл с именем, совпадающим с именем тесткейса в той же папке, где находится файл скрипта с этим тесткейсом. Кроме того, к имени файла в конце прибавляется текущая дата и время для того, чтобы в случае возникновения нескольких ошибок в одном тесткейсе файл со скриншотом не перезаписывался, а создавался новый файл с уникальным именем. Функция возвращает полное имя созданного файла. Пусть вас пока не смущает ключевое слово **withoptions** и вызов функции **BindAgentOption()**. Они будут рассмотрены позднее в разделе 10.3 Динамическое изменение настроек Агента, а сейчас мы лишь скажем, что SilkTest позволяет указать, какое количество одинаковых скриншотов должно быть сделано для того, чтобы захват изображения считался успешным, а в противном случае сгенерируется ошибка и изображение захвачено не будет. Здесь количество успешных захватов устанавливается равным единице, то есть захват изображения произойдет с первого раза.
    Теперь, для того, чтобы каждый раз не передавать второй параметр для функции **LogError()** для открытия скриншота (что придется делать достаточно часто), мы напишем новую функцию **Error()**, которая будет делать это автоматически. Выглядеть она будет так:

**Code**

```
[-] void Error (STRING sMsg)
        [ ] LogError (sMsg, "mspaint.exe {CreateScreenShot ()}")
```

11. Теперь, если вы захотите, чтобы во время прочтения сообщения об ошибке пользователь мог при одном нажатии на кнопку мыши посмотреть снимок экрана в момент возникновения ошибки, используйте функцию **Error()** вместо **LogError()**. Эти функции находятся в файле **TestAppUtils.inc**. Естественно, что вы можете указать любой просмотрщик картинок, который вам нравится, вместо MS Paint. Единственным недостатком этой функции состоит в том, что каждый **bmp-файл** имеет достаточно большой объем, отчего свободное место на жестком диске может уменьшаться достаточно быстро. В качестве решения этой проблемы можно использовать какую-либо утилиту, которая конвертирует **bmp-файлы** в файлы более экономного типа (например, **JPG**), однако мы оставим эту задачу на рассмотрение читателю.
12. **AppError(sMsg)** - прерывает выполнение тесткейса и передает управление рекавери системе (подробно об этом читайте в главе 3. Recovery-система). Кроме того в файл результатов будет выведено сообщение sMsg, в начале которого будет строка **"*** Error:"**
13. **RaiseError(iExcept, sMessage)** - генерирует исключение с заданным номером **iExcept** и указанным сообщением **sMessage** в файле результатов, также вписывая в

начале строку **"*** Error:"**. Подробнее об исключениях читайте в главе 10.4 Обработка исключительных ситуаций

14. **Verify (aActual, aExpected [, sDesc])** - сравнивает два значения **aActual и aExpected** любого типа и генерирует исключение, если они не совпадают. Оба параметра должны быть одного типа, иначе сгенерируется другое исключение **Type mismatch** (несовпадение типов)
15. **LogVerifyError (aData1, aData2 [, sDesc])** - выдает сообщение об ошибке несовпадения двух значений **aData1 и aData2**. В отличие от функции **Verify**, эта функция не проверяет, совпадают ли значения, а просто пишет сообщение об ошибке. Полезна в том случае, если функция **Verify** по каким-либо причинам не может быть использована
16. **ResPrintList (sName, lSubItems)** - открывает новый иерархический раздел в файле результатов с именем **sName** и заносит в него данные из списка **lSubItems**. Для того, чтобы просмотреть этот список, необходимо щелкнуть по значку [+] слева от названия списка **sName**.
17. **ResOpenList (sName), ResCloseList ()** - соответственно открывает и закрывает новый иерархический раздел **sName** в файле результатов. В отличие от функции **ResPrintList** содержимое иерархического раздела не передается через параметр функции. Вместо этого после открытия раздела с помощью функции **ResOpenList** выводе всех функций (**Print, LogError, LogWarning** и т.п.) будет производиться в этот раздел и закончится после того, как раздел будет закрыт с помощью функции **ResCloseList**. Допускается использование вложенных разделов
18. **SetAgentTrace (bTraceOn)** - устанавливает или снимает вывод в файл результатов вызовы функций и методов агентом. Аналогична установке или снятию флажка **Print Agent Calls** в окне **Runtime Options**. Возвращает предыдущее значение этого же параметра
19. **ResExport, ResExtract** - конвертируют файл результатов (.res) в формат экспортированного файла результатов (.rex) и текстовый файл (.txt) соответственно. Аналогичны операциям, осуществляемым через меню *Results - Export* и *Results - Extract* соответственно. Здесь же упомянем о функции **ResExportOnClose**, которая позволяет автоматически сконвертировать результаты в формат экспортированных результатов после завершения работы скрипта или тестплана.

# 8. Использование расширений (ActiveX, Java, .NET, Explorer extensions)

Если вы только начинаете работать с SilkTest-том и у вас нет необходимости работать с ActiveX-элементами или Java-приложениями, то данный раздел можно пропустить. В нем описываются особенности работы именно с приложениями, написанными на Java, либо использующих ActiveX-элементы. Кроме того, немного внимания уделено другим расширениям (или надстройками), идущим в поставке с SilkTest-ом.

Однако прежде, чем подключать надстройки, необходимо знать следующее: в SilkTest-е существует понятие **target machine** и **host machine**. **Host machine** - это компьютер, на котором запускается на выполнение скрипт в SilkTest-е. **Target machine** - это компьютер, на котором этот скрипт будет выполняться. В простом случае скрипт запускается и исполняется на одном и том же компьютере. В этом случае данный компьютер является и target и host machine одновременно. Однако можно сделать так, чтобы скрипт, запущенный на одном компьютере, выполнялся на другом, или даже на нескольких одновременно (подробнее об этом сказано в пункте 5. Распределенное, параллельное выполнение скриптов. Multitestcase). Для того чтобы надстройки включились, необходимо их настроить как на target, так и на host компьютерах.

Для того, чтобы подключить необходимые расширения на host machine надо зайти в меню *Options - Extensions*. Для подключения настроек для target machine необходимо зайти в *Пуск - Программы - SilkTest - Extension enabler*.

## 8.1 Работа с ActiveX-элементами

Что же собой представляет ActiveX? Это достаточно сложный вопрос, однако нам в рамках нашей задачи достаточно знать, что ActiveX-элементы - это такие же элементы управления, как и стандартные элементы; они имеют свои свойства и методы, с которыми можно точно так же работать: вызывать методы, присваивать свойствам значения и т.д. Точно так же, как у разных элементов управления существуют свои свойства и методы, ActiveX-элемент каждого типа тоже имеет свои собственные методы и свойства. В приложении Test Application, идущим в поставке с SilkTest-ом, нет такого элемента управления, поэтому мы рассмотрим работу с ним на примере простого приложения в MS Access (для дальнейшей работы у вас должен быть установлен Microsoft Office 2000 или выше).

Для этого мы откроем Access, создадим новую базу данных, затем перейдем в раздел Формы и создадим новую форму. После чего выберем пункт меню *Вставка - Элемент ActiveX*, после чего в списке выберем пункт **"Элемент управления Календарь"** и нажмем **ОК**. Теперь сохраним сделанные изменения и приступим к собственно изучению методов работы с ActiveX-элементами в SilkTest-е (описываемая база находится в прилагаемом архиве, см. файл **ActiveX.mdb**).

Прежде всего необходимо подключить расширения, причем, как уже было сказано ранее, сделать это для **host** и **target machine**.
В SilkTest-е выберем пункт меню *Options - Extensions*. Откроется список подключенных расширений. Нажмем кнопку *New* и впишем имя exe-файла нашего приложения (или нажмем кнопку *Browse*, выберем необходимый файл и нажмем *Открыть*). Для нашего примера необходимо выбрать файл **msaccess.exe**, который находится в папке **\Program**

**Files\Microsoft Office\Office10** (в зависимости от версии офиса имя последней папки может иметь разные цифры в названии). После этого нажмем кнопку **ОК**. Имя файла появится в списке расширений. Теперь необходимо включить флажок **ActiveX** напротив нашего файла (средний флажок справа от названия файла) и нажать **ОК**. Расширения для **host machine** включены. Теперь необходимо повторить те же самые действия для **target machine**. Повторите все действия, описанные выше, только вместо пункта меню *Options - Extensions* выберите *Пуск - Программы - SilkTest - Extension Enabler*.

Если вы все сделали правильно, то у вас должен получиться результат, показанный на рисунке 8.1.

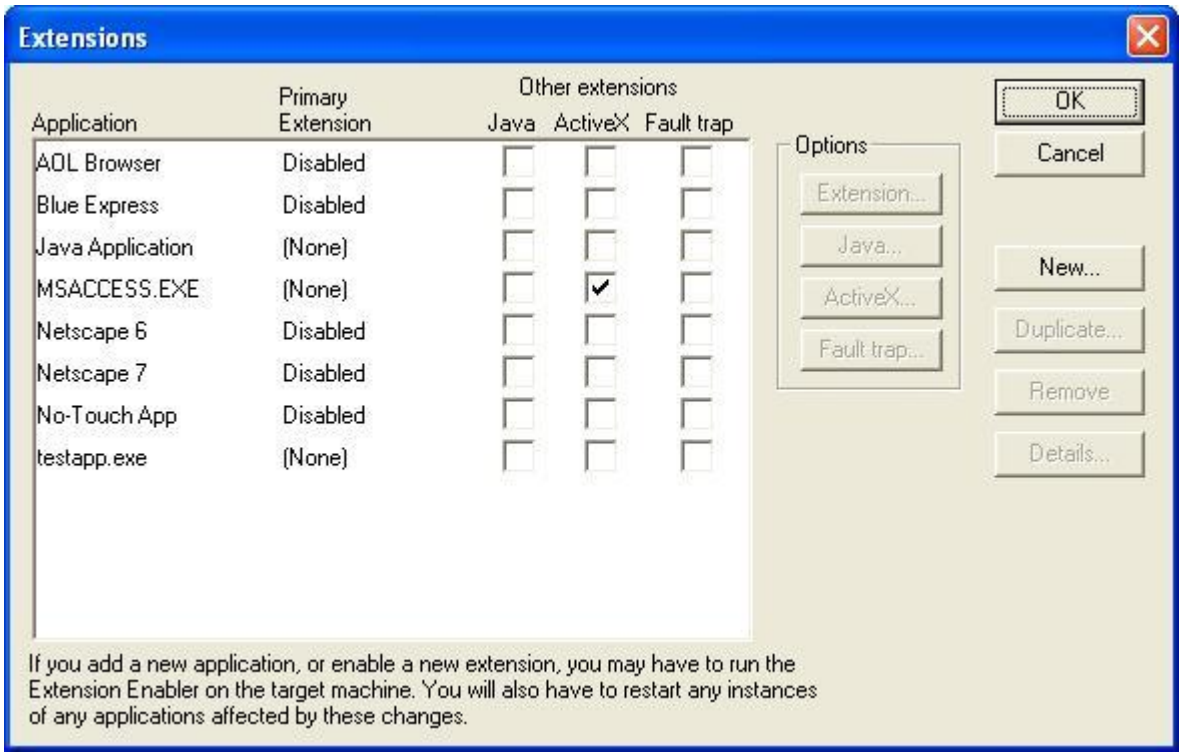



Рисунок 8.1.

Примечание. В некоторых случаях файлы, в которых хранятся настройки о подключаемых расширениях, блокируются системой или самим SilkTest-ом и после описанных выше действий никакие изменения не добавляются (проверить это можно, повторно открыв расширения и проверив, что все осталось так, как мы и задумывали). Если такое случилось, достаточно перезагрузить систему и выполнить эти действия еще раз.

После подключения расширений необходимо перезапустить приложение, для которого подключались расширения (в данном случае это Microsoft Access).

Теперь нам необходимо записать объявление нужного нам класса. Откроем форму, в которую помещен календарь (в прилагаемой базе она открывается автоматически при открытии файла), затем в SilkTest-е выберем пункт меню *Record - Class*, наведем курсор мыши на календарь в форме и нажмем *Ctrl-Alt*. Если все предварительные настройки были сделаны корректно, то в окне Record Class вы увидите список методов и свойств, которые определены для этого ActiveX-элемента. Если же списки пусты, значит расширение ActiveX не было активировано для данного приложения либо этот элемент не является таковым.

Вставим описание нового класса (кнопка *Paste to Editor*). Теперь опишем наше окно Microsoft Access через меню *Record - Window Declaration*. Теперь при наведении курсора

мыши на календарь SilkTest правильно определяет его класс - **OLECalendar**. Если бы мы сделали это до подключения расширений, то класс этого элемента был бы **CustomWin**, и работать с ним нормально было бы невозможно.

Теперь посмотрим на записанные свойства и методы. В принципе их назначение понятно интуитивно. Например: **iMonth** - месяц, **iYear** - год. Метод **Today()** устанавливает в календаре текущую дату. Методы **NextDay**, **NextWeek**, **NextMonth** и **NextYear** устанавливают соответственно следующий день, неделю, месяц и год.

Посмотрите пример тесткейса, работающего с этим элементом (**Test_ActiveX** из файла **TestApp.t**). В качестве самостоятельного упражнения попробуйте написать тесткейс, который устанавливает в календаре завтрашнюю дату и затем проверяет правильность установленной даты (подсказка: используйте функцию **AddDateTime()** ).

Еще некоторые замечания по использованию ActiveX-расширений:

1. в случае, если тесткейс остановился, выдав ошибку **"Property ... not found"** или **"Call to object failed"**, - скорее всего, сбились настройки ActiveX-а. В этом случае рекомендуется закрыть тестируемое приложение и Агента SilkTest-а и затем перезапустить тесткейс.
2. никогда не закрывайте приложение, для которого включена поддержка ActiveX-ов, нестандартным методом (например, через *Task Manager*). В этом случае ошибки, описанные в предыдущем пункте, будут возникать очень часто.

## 8.2 Работа с .NET-приложениями

Процесс подключения расширений для .NET-приложений выглядит так же, как и для приложений с ActiveX-элементами, за исключением того, что вместо включения флажка ActiveX в окне **Extensions** необходимо выбрать в выпадающем списке **Primary Extension** пункт **.NET Ext** (рис. 8.2).

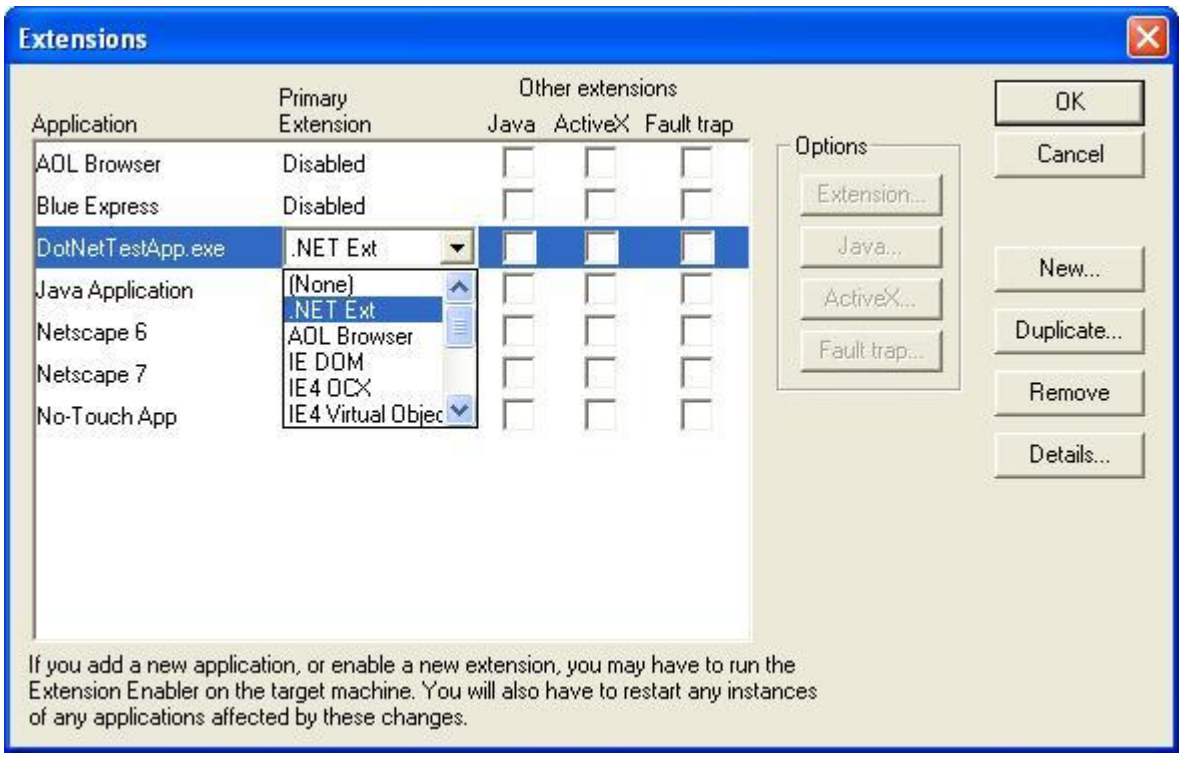



Рисунок 8.2.

После того, как вы сделаете это для **target** и **host** машин, SilkTest автоматически добавит файл dotnet.inc в список подключаемых файлов (чтобы увидеть этот список выберите пункт меню *Options - Runtime*, список находится в поле **Use Files**). Там же вы можете добавлять любые файлы, которые хотите чтобы подключались автоматически при открытии SilkTest-а.

Если отрыть этот файл (он находится в папке *"C:\Program Files\Segue\SilkTest\Extend"*), то будет видно, что в нем определены классы для многих .NET-объектов и окон, которые наследуются от стандартных классов. Отличие состоит лишь в префиксе **Swf**, характерном для .NET-классов. В остальном же работа с ними не отличается от стандартных классов.

Кроме того, отличие .NET-объектов от обычных в том, что каждый из них похож на ActiveX-контрол, то есть вы можете, используя пункт меню *Record - Class* посмотреть список доступных свойств и методов для любого объекта. Это удобно в случае, если какой-либо объект не является стандартным, а разработан специально для данного приложения. Достаточно записать все его свойства и методы, как и для ActiveX-элементов, и работать с ними.

В тестовом приложении DotNetTestApp.exe, поставляемом с SilkTest-ом, таких элементов нет, поэтому мы воспользуемся одним из стандартных, а именно **CheckedListBox**. В качестве примера рассмотрим работу со списком чекбоксов, каждый элемент которого может находиться во включенном либо выключенном состоянии.

Прежде всего, необходимо закомментировать объявление этого класса в файле **dotnet.inc** (иначе SilkTest не позволит добавить его объявление). После чего откроем это окно (в **DotNet Test Application** пункт меню *Control - CheckedListBox*), а в SilkTest-е выберем пункт меню *Record - Class*, наведем курсор мыши на список и нажмем сочетание клавиш *Ctrl-Alt*. Если расширения были подключены правильно, то в окне **Record Class** мы увидим список доступных для этого списка свойств и методов (см. рис. 6.3).

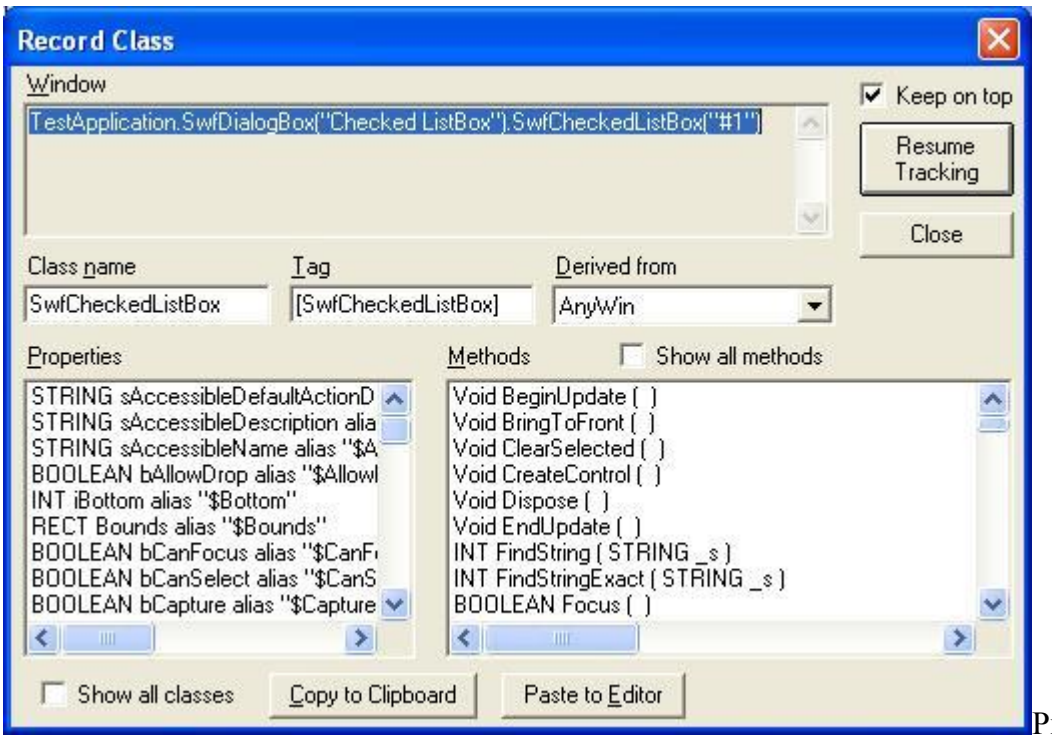


Рисунок 8.3.

Примечание: так как для .NET-приложений используются классы, отличные от стандартных классов, необходимо записать декларации окон заново иначе. В файле **TestAppNet.inc** главное окно приложения называется **wTestAppNet**. Там же находится описание свойств и методов необходимых классов.

Теперь, когда у нас есть список свойств и методов, мы можем их использовать по своему усмотрению. Для примера посмотрите тесткейс **Test_CheckedListBox** в файле **TestAppNet.inc**. Этот тесткейс демонстрирует использование трех методов класса **SwfCheckedListBox**: **get_Sorted**, **SetItemChecked** и **FindString.**

В качестве упражнения попробуйте написать тесткейсы с использованием методов одного из классов: **ProgressBar, PictureBox, DataGrid, LinkLabel** или **MonthCalendar**, которые представлены в приложении **NET Test Application.**

Некоторые замечания:

1. Если во время записи методов класса в окне **Record Class** включить опцию **Show all methods**, то среди записанных методов могут оказаться методы, которые начинаются со строки **"///**"** (три символа слеш и две звездочки). Эти методы не могут быть вызваны средствами SilkTest-а.
2. Если некоторые из объектов, которые используются в тестируемом приложении, работают некорректно, возможно придется воспользоваться **ClassMap'ом**, как это объясняется в главе 1.4 Использование Class Map и расширение встроенных классов

## 8.3 Тестирование Java-приложений

При тестировании Java-приложений вам также необходимо подключить расширения, иначе SilkTest не будет "видеть" ни одного элемента управления в окнах. Прежде всего убедитесь, что в окнах **Extension Enabler** для **target** и **host** машин включена поддержка Java для Java-приложений (рис. 8.4).

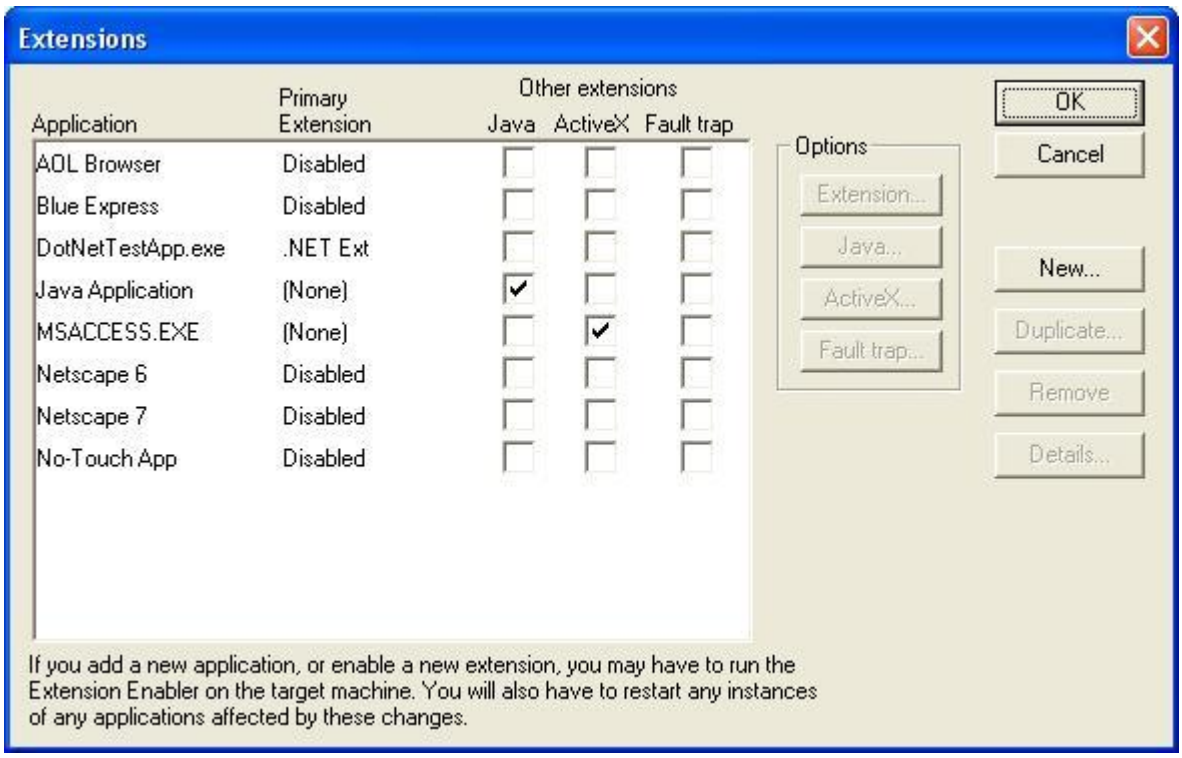



Рисунок 8.4.

Если опция **Java Application** есть, но галочка Java не включена - включите ее. Если же такой опции нет - создайте новое правило, по аналогии с включением поддержки ActiveX, однако в поле, где вписываются имена файлов (окно **Extension application**) впишите следующую строку:
**java.exe, jre.exe, jrew.exe, appletviewer.exe, javaw.exe** После этого опция **Java Application** появится в списке. Для нее необходимо включить опцию Java.

После этого, как и в случае с .NET-приложениями, в список **Use Files** добавится файл JavaEx.inc

Теперь необходимо настроить SilkTest таким образом, чтобы он правильно работал с Java-приложениями. Для этого запустите тестируемое приложение (в нашем случае это "JFC Test Application for JDK 1.2 and above"), в SilkTest-е выберите пункт меню *Tools - Extension Enabler* и в списке приложений выберите нужное приложение, после чего нажмите кнопку Select (рис. 8.5).

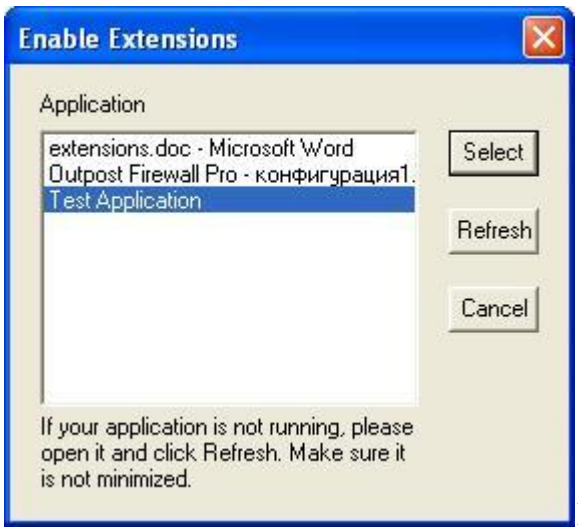


Рисунок 8.5.

После этого SilkTest выдаст сообщение о том, что необходимо скопировать файл **SilkTest_Java3.jar** в папку **\lib\ext** приложения, после чего предложит перезапустить приложение и нажать кнопку Test.

Если после всех вышеописанных действий вы получили сообщение об успешной конфигурации (окно с заголовком **Test Passed**), то вы успешно настроили SilkTest для работы с Java-приложением. В случае сообщения об ошибке вам, возможно, придется вручную скопировать указанный SilkTest-ом файл в папку **\lib\ext**, которая находится в папке, куда было установлено тестируемое приложение (если такой папки нет - создайте ее).

Примечание: если вы используете на компьютере какую-либо программу-фаервол (firewall, например **Outpost Firewall, BlackIce** и т.д.), вам понадобится добавить файл java.exe в список доверенных приложений, или отказаться от его использования. В противном случае сконфигурировать SilkTest для работы с Java-приложениями не получится.

Если у вас нет Java-приложения и вы хотите просто потренироваться на примерах, которые поставляются в SilkTest-ом, вам необходимо:

1. установить **Java 2 SDK Standard Edition** любой версии (при написании этой книги использовалась версия 1.4.2.05); скачать дистрибутив можно на сайте Sun Microsystems
2. найти в папке **"C:\Program Files\Segue\SilkTest\JavaEx"** все **bat-файлы**, предназначенные для запуска Java-приложений (**AWT_TestApplication.bat, JFC_TestApplication.bat, Swing11TestApp.bat**), открыть их в любом редакторе и в строку **"set JavaRun="** в конце дописать путь, по которому после установки Java 2 SDK у вас находится файл java.exe. Например:
   **set JavaRun=C:\j2sdk1.4.2_05\bin**

Теперь можно конфигурировать SilkTest, как было сказано выше.

В файле **TestAppJava.inc** имеется описание окна **wTestAppJava** и диалогового окна **dListBox**. В файле **TestAppJava.t** - простой тесткейс, работающий с Java-приложением, добавляющий несколько элементов в **ListBox**, а затем проверяющий, правильно ли они добавились.

Для Java-элементов, точно так же, как и для .NET и ActiveX элементов, можно использовать их встроенные свойства и методы. Просмотреть и записать их можно, используя меню *Record - Class* в SilkTest-е.

## 8.4 Расширения Explorer'a

Собственно о WEB-тестировании с использованием браузера **Internet Explorer**, подробно рассказывается в главе 4. Тестирование WEB-приложений, поэтому здесь мы не будем на этом останавливаться, а расскажем о некоторых других особенностях и возможностях.

Кроме WEB-тестирования, где скрипт постоянно работает с окном браузера, подобные компоненты **Internet Explorer'a** могут использоваться и в других приложениях, в том числе и в других браузерах, например, для выведения отформатированных отчетов на экран. В SilkTest-е реализована поддержка двух браузеров: **Internet Explorer и Netscape Navigator** (в SilkTest версии 8.0 включена также поддержка **Mozilla Firefox**).

Для того чтобы вручную подключить расширения для **Internet Explorer'a**, необходимо добавить в список **Extension Enabler'a** файл **iexplore.exe** (как это сделать, подробно описывается в главе 8.1 Работа с ActiveX-элементами), и установить для него **Primary Extension** равным **IE DOM**, после чего перестартовать браузер.

Для подключения расширений **Netscape** необходимо для файла **netscp.exe** подключить **Primary Extension Netscape DOM**.

Если ваше приложение использует вывод на экран каких-либо отчетов в формате **HTML**, то, возможно, для этого используется компонент **Internet Explorer'a**. Для того, чтобы иметь возможность работать с такими отчетами, необходимо также для ***.exe** файла этого приложения подключить одно из расширений **Internet Explorer'a: IE DOM, IE OCX** либо **IE Virtual Objects**, подобно тому, как мы подключали ActiveX для приложения MS Access. В противном случае вы будете иметь возможность работать с содержимым странички только как с обычным текстом, используя буфер обмена.

Кроме того, существует также несколько браузеров, которые не являются браузерами как таковыми (например, **MyIE, Avant Browser**). Для того чтобы работать с ними, как с браузером, необходимо для них подключать расширения других браузеров. Например, для **MyIE** необходимо подключить расширение **IE4 OCX**, а для **Avant Browser'a - IE DOM**. После этого все объекты внутри HTML-страничек будут правильно распознаваться SilkTest-ом.

# 9. Использование внешних данных (Data-driven test)

SilkTest позволяет, помимо объявления переменных в тесткейсах, использовать данные, которые считываются из какого-либо внешнего источника (базы данных, текстовых файлов в определенном формате, электронной таблицы). Тесткейсы, считывающие данные из внешнего источника, называются **Data Driven Testcases** (дословно: тесткейсы, управляемые данными). Такой тесткейс можно рассматривать как некий шаблон, который можно использовать несколько раз, каждый раз работая с новыми данными.

SilkTest обладает широкими встроенными возможностями для работы с базами данных (работа с текстовыми файлами и электронными таблицами в данном случае аналогична работе с базой данных). Прежде всего выясним, в каких случаях полезны эти возможности:

- Для хранения данных, которые используются разными скриптами, однако определяются один раз. К таким данным можно отнести, например, константы, используемые в тесткейсах, данные из тестируемого приложения, которые используются в различных частях этого же приложения и т.п.
- Для хранения данных, которые необходимо ввести в приложение перед началом тестирования. Например, прежде, чем начать запуск тесткейсов, необходимо добавить в тестируемое приложение записи, с которыми затем будут работать скрипты
- Для непосредственного доступа к базе данных тестируемого приложения. Например, для проверки, что по определенному запросу в приложении выводится необходимый набор данных, можно считать эти данные из базы данных приложения, а затем сравнить с тем, что выводится в приложении
- Для хранения результатов запущенных тесткейсов. Так, вместо того, чтобы использовать файлы результатов SilkTest-а, можно выводить данные в базу данных, где с ними будет удобно работать (находить среднее время работы тесткейсов, отлавливать шаги, на которые затрачивается наибольшее время и т.п.)
- При наличии хорошо отлаженного и используемого тестдрайва (среды, управляющей запуском тесткейсов, например запускающей автоматически наборы скриптов в определенное время без непосредственного участия человека) в базе можно хранить наборы тесткейсов, предназначенные для различных видов тестирования, или для тестирования разных частей приложения

Подобное хранение данных в одном месте, откуда они будут доступны всем, хорошо тем, что константы, определения и т.д. не раскиданы по разным файлам, что затрудняет навигацию между ними и их поиск, а находятся рядом, что упрощает их просмотр и редактирование. Кроме того, если необходимо ввести новые константы, то не возникает вопроса, где их хранить, так как все хранится в одной базе.

Теперь рассмотрим функции, предназначенные для работы с базами данных.

- **DB_Connect (con_string)** - создает подключение к базе данных; в качестве параметра необходимо передать строку подключения con_string, в которой указывается идентификатор базы данных и все необходимые дополнительные параметры. Возвращает указатель (handle) типа **HDATABASE**. Например, для подключения к таблице **Excel** строка подключения будет иметь следующий вид:

- 
- `HDATABASE hDB = DB_Connect ("DSN=Segue DDA Excel;DBQ=C:\testxls.xls)`

  Здесь необходимо сделать небольшое отступление. Строка "DSN=Segue DDA Excel" указывает SilkTest-у строку подключения системного DSN (Data Source Name - системный источник данных), который устанавливается в системе при установке SilkTest-а. Вы можете создать свой собственный источник данных, который будет связан с установленными драйверами **ODBC**. Для этого необходимо зайти в *Пуск - Настройка - Панель управления - Администрирование - Источники данных ODBC*, перейти на вкладку **Системный DSN**, нажать кнопку **Добавить,** выбрать соответствующий драйвер (в нашем случае для работы с таблицами **Excel** необходимо выбрать элемент **Driver to Microsoft Excel**) и указать его имя. После этого вместо строки **Segue DDA Excel** можно указывать имя, заданное в конце этих операций. Кроме того, можно также использовать источники данных, перечисленные на вкладке **"Пользовательский DSN"** окна **"Администратор ODBC"**, однако их имена отличаются в русской и английской версиях **MS Office**, поэтому могут возникнуть проблемы при переносе скриптов с одной системы на другую.

- **DB_Disconnect (hdbc)** - отключает SilkTest от ранее присоединенного источника данных, освобождая все ресурсы. Единственный параметр - полученный ранее указатель на базу данных. Например, следующий пример подключается к базе данных **c:\testxls.xls** и сразу же отключается от нее:
- 
- `[ ] HDATABASE hDB = DB_Connect ("DSN= Segue DDA Excel;DBQ=C:\testxls.xls")`
- `[ ] DB_Disconnect (hDB)`
- **DB_ExecuteSql (hdbc, sql_stmnt)** - выполняет запрос SQL, переданный строковым параметром **sql_stmnt** для базы **hdbc**. Например, следующий код создаст таблицу test_tbl с двумя полями - **Name и Age**:
- 
- `DB_ExecuteSQL (hDB, "CREATE TABLE test_tbl (Name char, Age int)")`

  А следующий код добавит в созданную таблицу новую запись:

  ```
  DB_ExecuteSQL (hDB, "INSERT INTO test_tbl (Name, Age) VALUES ('Petrov Petr Petrovich', 25)")
  ```

- **DB_FinishSql (hstmnt)** - удаляет результат, который был получен с помощью функции **DB_ExecuteSQL**. После использования этой функции невозможно использовать функции **DB_FetchNext** и **DB_FetchPrev** (см. ниже). Кроме того, для того, чтобы удалить все результаты типа **HSQL**, можно воспользоваться функцией **DB_Disconnect**, которая освободит все использовавшиеся ресурсы **HSQL**
- **DB_FetchNext (hstmnt, ...)** - считывает следующую строку из таблицы. Параметр **hstmnt** - результат, полученный с помощью функции **DB_ExecuteSQL**. Остальные параметры этой функции - переменные, в которые будут считываться данные из столбцов таблицы. Возвращает **FALSE** в том Например, для считывания данных из таблицы **test_tbl**, которую мы создали в предыдущем примере, можно написать следующее:
- 
- `[ ] hSql = DB_ExecuteSql (hDB, "SELECT * FROM [test_tbl$]")`
- `[-] while (DB_FetchNext (hSql, s1, i))`

- `[ ] Print ("Name = {s1}, Age = {i}")`

  Запись "[test_tbl$]" в предложении **FROM** сделана так для того, чтобы захватить все строки таблицы, а не только строки именованного диапазона, созданного при использовании инструкции **INSERT**. В этот диапазон будет входить только первая запись. Для того, чтобы посмотреть список имеющихся именованных диапазонов, вы можете в **MS Excel'e** зайти в меню Вставка - Имя - Присвоить и просмотреть их. При обращении непосредственно к таблице, используя символ $ в конце имени (как это показано в нашем примере) поиск будет осуществляться по всем строкам, пока не встретится первая пустая строка

- **DB_FetchPrev** - аналогична предыдущей функции, с той лишь разницей, что переходит не к следующей строке, а к предыдущей
- **DB_Columns (hdbc, catalog-name, schema-name, table-name, column-name)** - возвращает список полей таблицы. Тип возвращаемого значения, как и для функций **DB_Fetch... - HSQL**. Параметры **catalog-name, schema-name, table-name, column-name** задают соответственно имена каталога, схемы, таблицы и колонки и могут иметь вид регулярных выражений. Допускаемые символы замены: "%" (знак процента) - замена нескольких символов, "_" (знак подчеркивания) - один любой символ. Если необходимо пропустить какой-либо из параметров, необходимо присвоить ему значение **NULL**. Например, если необходимо выбрать все колонки, начинающиеся с символа N из созданной нами таблицы, мы можем использовать следующий пример:
-
- `[ ] STRING sTbl = "test_tbl"`
- `[ ] STRING sSchema = NULL`
- `[ ] STRING sCatalog = NULL`
- `[ ] STRING sColumn = "N%"`
- `[ ] HSQL hSql`
-
- `[ ] HDATABASE hDB = DB_Connect ("DSN=Segue DDA Excel;DBQ=C:\testxls.xls; READONLY=FALSE")`
- `[ ] hSql = DB_Columns (hDB, sCatalog, sSchema, sTbl, sColumn)`
- `[-] while (DB_FetchNext (hSql, sCatalog, sSchema, sTbl, sColumn))`
- `[ ] Print (sColumn)`
- `[ ] DB_Disconnect (hDB)`
- **DB_Tables (hdbc, catalog-name, schema-name, table-name, table-type)** - возвращает список таблиц базы данных. Кроме того, последним параметром можно указать тип таблиц, которые необходимо просматривать ("table", "view" и т.д.)
- **DB_Procedures (hdbc, catalog-name, schema-name, proc-name)** - возвращает список процедур базы данных
- **DB_PrimaryKeys (hdbc, catalog-name, schema-name, table-name)** - возвращает список ключевых полей таблицы. Обратите внимание, что эта функция не возвращает список полей для нескольких таблиц в одном обращении, поэтому в этой функции не допускается использование символов групповой замены для параметра **table-name**
- **DB_ForeignKeys (hdbc, pcatalog-name, pschema-name, ptable-name, fcatalog-name, fschema-name, ftable-name)** - возвращает список связанных полей. В качестве параметров передаются ключевые имена (p...-name) и связанные (f...-name) имена. В именах таблиц также не должно содержаться символов групповой замены

Это неполный список функций для работы с базами данных. Предоставляемый SilkTest-ом. Полный список с более подробным описанием можно найти в справке по SilkTest-у в разделе *4Test Reference - Functions - Database*.

Если вы часто работаете с какой-либо базой данных, скорее всего, будет удобнее написать свой набор функций, или оформить их в виде класса с необходимыми методами, чтобы иметь возможность считывать данные из любой строки и любого столбца, не используя каждый раз функции **DB_Fetch...**, или передавать в качестве параметра лишь путь к базе данных, не используя напрямую переменные типа **HSQL**.

# 10. Возможности языка 4Test

Так как язык 4Test очень похож на язык c++, то мы не будем останавливаться на рассмотрении всех возможностей этого языка, а лишь укажем на некоторые особенности, которые могут быть полезны в работе.

## 10.1 Циклы for

В SilkTest-е существует 3 вида цикла for:

- стиль языка C
- стиль языка Basic
- цикл for...each

Каждый из них рекомендуется применять в разных ситуациях.

Например, если вам необходимо выполнить какое-либо действие для всех элементов списка, размер которого неизвестен заранее (например, он был получен динамически из элемента управления ListView), то лучше применить второй вариант, так как он один раз вычислит размер списка.

Однако, если при этом количество элементов в списке меняется внутри цикла, то воспользоваться лучше первым вариантом, так как при этом размер списка будет пересчитываться каждый раз при входе в цикл. Третий вид цикла for рекомендуется применять для списков с однотипными элементами.

Примеры:

**Code**

```
[-] for i = 1 to ListCount (li)
        [-] if (li[i] < 5)
                [ ] ListDelete (li, i)
[ ] Print (li)
```

При выполнении данного кода возникнет исключение:

**Code**

```
*** Error: Index value of 8 exceeds list size of 7
```

Для того чтобы этого не случилось, необходимо переписать его следующим образом:

**Code**

```
[-] for (i = 1; i <= ListCount (li); i++)
        [-] if (li[i] < 5)
                [ ] ListDelete (li, i)
```

```
                [ ] i--
[ ] Print (li)
```

Пример использования цикла for...each можно посмотреть в главе 2

## 10.2 Конкатенация строк

Кроме привычного способа конкатенации (сложения строк) с помощью оператора "+" SilkTest предлагает еще один способ: внутри строковой константы можно указать имя переменной, обрамленной символами { и }. Преимущество такого способа в том, что кроме строковых переменных можно использовать переменные любых других типов, их приведение к строке произведется вручную. Кроме того, этот способ более простой и проще читается.

Предположим, нам необходимо соединить в одну строку переменные нескольких типов и разделить их внутри пробелами. Сравните обычный способ:

**Code**

```
[ ] STRING s = "Sample string"
[ ] INTEGER i = 138
[ ] REAL r = 34.786
[ ] BOOLEAN b = TRUE
[ ]
[ ] STRING sResult = s + " " + Str (i) + " " + Str (r) + " " + [STRING]b
[ ] Print (sResult)
```

с вариантом, который доступен в SilkTest-е (здесь показан уже лишь процесс объединения всех переменных, объявления переменных такие же, как в предыдущем примере):

**Code**

```
[ ] STRING sResult = "{s} {i} {r} {b}"
```

Очевидно, что второй вариант гораздо проще писать, читать и редактировать.

## 10.3 Динамическое изменение настроек Агента

Во время работы скрипта иногда необходимо на время изменить параметры агента. Например, мы это делали в главе 7. Обработка результатов (Results) для при создании скриншота. Теперь рассмотрим эту возможность подробнее.

С помощью изменений настроек Агента можно изменить многие параметры, которые заданы в **opt**-файле. Также эти параметры задаются в окне **Agent Options** (Options - Agent). Например, с помощью опций Агента можно изменить время ожидания окна SilkTest-ом, время задержки при вводе данных с клавиатуры или с помощью мыши, проверку на активность объекта при работе с ним, либо проверку на то, что объект доступен для работы (enabled) и многое другое. Все параметры Агента, доступные для изменения, перечислены в справке по SilkTest-у. Кроме того, они отображаются внизу окна **Agent Options** при их изменении. Для работы с настройками Агента существует

класс **Agent**. Методы этого класса, предназначенные для работы с настройками, перечислены ниже.

- **Agent.GetOption (Option)** - возвращает значение определенной опции. Тип возвращаемого значения зависит от опции
- **Agent.SetOption (Option, aValue)** - устанавливает значение опции. Возвращает предыдущее значение этой же опции
- **Agent.IsOptionValid (aoOptionName)** - проверяет, является ли опция допустимой

Метод **Agent.SetOption** устанавливает значение опции на все время работы скрипта. То есть при изменении какой-либо настройки во время работы одного тесткейса, эти же настройки будут сохранены и для всех остальных тесткейсов до тех пор, пока не будут изменены явно.

Если же необходимо изменить настройки временно для выполнения небольшой части кода, то необходимо пользоваться ключевым словом **withoptions** и функцией **BindAgentOption.** В данном случае после выхода из блока **withoptions** все изменения, сделанные в нем с помощью функции **BindAgentOption**, будут возвращены обратно.

Приведем пример. В SilkTest-е по умолчанию включена опция "Verify that windows are active". Это значит, что вы не сможете работать с объектами неактивного окна и следующий пример вызовет ошибку.

**Code**

```
[-] testcase Test_Agent_Options () appstate apsTestApp
        [ ]
        [ ] wTestApp.SetActive ()
        [ ] wTestApp.Control.ListBox.Pick ()
        [-] if (!dListBox.Exists ())
                [ ] LogError ("ListBox dialog not opened")
        [ ] dListBox.btnPopup.Click ()
        [ ] Sleep (0.5)
        [-] if (!dMsg.Exists ())
                [ ] LogError ("Message Box not opened")
        [ ] Print (dListBox.lstItemType.sValue)
```

Результат работы:

**Code**

```
        [ ] *** Error: Window '[PopupList]Item type:' is not active
        [ ] Occurred in dListBox.lstItemType.sValue.Get
```

Как видно из примера, SilkTest не смог взять значение свойства **sValue** из списка, который находится в неактивном окне. Переделаем этот пример, чтобы сделать его рабочим, независимо от того, активно окно или нет (мы приведе здесь лишь ту часть скрипта, которая была изменена):

**Code**

```
[-] withoptions
        [ ] BindAgentOption (OPT_VERIFY_ACTIVE, FALSE)
        [ ] BindAgentOption (OPT_VERIFY_ENABLED, FALSE)
        [ ] Print (dListBox.lstItemType.sValue)
```

Здесь мы выключаем проверку на то, что объект должен быть активным и доступным, после чего получаем результат, который нам был необходим.

## 10.4 Обработка исключительных ситуаций

Каждый раз при возникновении ошибки во время выполнения скрипта SilkTest генерирует исключение и выполнение скрипта останавливается. Таким же образом, в том числе, работаю функции проверки (**Verify, RaiseError** и т.д.). Однако не всегда во время возникновения ошибки необходимо прерывать работу скрипта. В частности, если проверка каких-либо данных не прошла успешно, вполне вероятно, что скрипт может выдать сообщение об ошибке и продолжать работать дальше.

Именно для таких целей в SilkTest-е существует возможность работать с исключениями.

Для того, чтобы перехватить исключение и не дать SilkTest-у остановить работу скрипта, необходимо поместить ту часть кода, где мы ожидаем появление ошибки, в блок **do...except**.

Возьмем самый просто пример деления на ноль:

**Code**

```
[ ] Print (1/0)
[ ] Print ("something")
```

При выполнении данной строки сгенерируется исключение:

**Code**

```
*** Error: Divide by zero
```

и строка "something" выведена в результатах не будет

Теперь поместим эту строку в блок **do...except** и в случае возникновения исключительной ситуации будем об этом сообщать в результатах:

**Code**

```
[-] do
        [ ] Print (1/0)
[-] except
        [ ] LogError ("Exception occured")
[ ] Print ("something")
```

Результат теперь примет такой вид:

**Code**

```
[ ] Exception occured
[ ] something
```

Теперь рассмотрим некоторые операции для работы с исключениями:

- **ExceptPrint ()** - функция выводит на печать информацию об ошибке
- **ExceptData ( )** - возвращает информацию об исключении
- **ExceptNum ( )** - возвращает номер исключения
- **ExceptLog ( )** - выводит на печать информацию об исключении. Отличие от первой функции в том, что информация выводится, как ошибка, а не как обычный комментарий
- **ExceptClear ()** - очищает информацию о текущем исключении
- **ExceptCalls ()** - возвращает список вызовов всех функций от тесткейса до функции, в которой произошла ошибка
- **raise [integer-expr [, expr [, cmd-line]]]** - генерирует исключение с заданным номером (**integer-expr**)
- **reraise** - генерирует исключение повторно

Например, если необходимо перехватить какое-либо конкретное исключение, но при этом другие исключения должны возникать, как и прежде, будет уместно написать следующее:

**Code**

```
[-] do
        [ ] Print (1/0)
[-] except
        [-] if ExceptNum () == E_DIVIDE_BY_ZERO
                [ ] Print ("Division by zero as expected")
        [-] else
                [ ] Print ("Unexpected exception: {ExceptNum ()}, {ExceptData
()}")
                [ ] reraise
```

В данном примере никакого другого исключения не будет, он приведен лишь для демонстрации перехвата исключительных ситуаций.

Функция **ExceptCalls** полезна в том случае, если вместо встроенных результатов SilkTest-а используется свой генератор отчетов. В этом случае можно в отчет поместить весь список вызовов функций и методов, вплоть до того, в котором произошло исключение.

Например, напишем 2 функции, вторая из которых будет генерировать исключение, а вызываться будет из первой. Первая же, в свою очередь, будет вызываться из функции **main**:

**Code**

```
[-] void First ()
```

```
        [ ] Second ()
[-] void Second ()
        [ ] raise E_DIVIDE_BY_ZERO, "Divide by zero generated"
[ ]
[-] main ()
        [ ] LIST OF CALL lcCalls
        [ ] CALL cCall
        [ ]
        [-] do
                [ ] First ()
        [-] except
                [ ] lcCalls = ExceptCalls ()
                [-] for each cCall in lcCalls
                        [ ] Print ("Error occured in module '{cCall.sModule}',
function '{cCall.sFunction}', line {cCall.iLine}")
                [ ] LogError ("{ExceptNum ()}, {ExceptData ()}")
```

Результат будет иметь такой примерно вид:

**Code**

```
[ ] Error occured in module 'TestApp.t', function 'Second', line 1516
[ ] Error occured in module 'TestApp.t', function 'First', line 1514
[ ] Error occured in module 'TestApp.t', function 'main', line 1523
[ ] -11500, Divide by zero generated
```

## 10.5 Преобразование типов

В процессе работы с данными в тесткейсах иногда возникает необходимость преобразования данных из одного типа в другой (например, использовать число как строку или наоборот, преобразовать запись в список, и т.д.).

В SilkTest-е существуют два типа преобразования данных: явный и неявный.

При явном преобразовании необходимо использовать оператор **[]** для указания типа, в который преобразуются данные. Например:

**Code**

```
STRING s = [STRING]55
```

Здесь производится явное преобразование из **INTEGER** в **STRING**.

Неявное преобразование осуществляется автоматически и не требует указания оператора **[]**. Например:

**Code**

```
        [ ] LIST OF STRING ls = {"one", "two", "three"}
        [ ] ARRAY OF STRING as = ls
```

Ниже дана таблица явных и неявных преобразований, возможных в SilkTest-е с примечаниями к некоторым из них.

Таблица 10.1 Явные и неявные преобразования типов в SilkTest-е

| Явные преобразования | Неявные преобразования |
|---|---|
| BOOLEAN –> INTEGER | INTEGER –> REAL |
| INTEGER –> enum | INTEGER –> BOOLEAN |
| BOOLEAN –> enum | **** REAL –> INTEGER |
| любой_не_строковый –> STRING | enum –> INTEGER |
| record –> LIST | LIST –> record |
| STRING –> INTEGER | LIST –> ARRAY |
| STRING –> GUITYPE | ARRAY –> LIST |
| ** DATETIME –> DATE | * LIST OF type-1 –> LIST OF type-2 |
| ** DATETIME –> TIME | * ARRAY of type-1 –> ARRAY OF type-2 |
| ** DATE –> DATETIME | GUITYPE –> INTEGER |
| ** TIME –> DATETIME | enum –> BOOLEAN |
| SET –> LIST | *** STRING –> DATETIME |
| SET –> INTEGER | STRING –> HMACHINE |
| INTEGER –> SET | LIST –> SET |
| INTEGER –> SEMAPHORE | |

Примечания: * Для элементов массива или списка используются правила преобразования встроенных типов ** Подробнее смотри описание типа DATETIME *** Если строка в формате ISO: YYYY-MM-DD HH.MM.SS.MSMSMS **** Преобразование REAL в INTEGER осуществляется с помощью отбрасывания дробной части (не округление!)

Обратите внимание, что при попытке преобразования строки в число SilkTest вернет не преобразованную в число строку, а код первого символа этой строки. Для корректного преобразования строки в число необходимо воспользоваться функцией **Val**. Например:

**Code**

```
[ ] INTEGER i = Val ("654")
[ ] REAL r = Val ("654.554")
```

# 11. Другие возможности

## 11.1 Работа с файловой системой

### 11.1.1 Работа с файлами

Зачастую при написании скриптов возникает необходимость считывать данные из внешних файлов или, наоборот, записывать туда какие-либо данные. В SilkTest-е существуют несколько функций, позволяющих работать с файлами:

1. **FileOpen (sPath, fmMode [, fsShare[, ftType]])** - открывает файл. Возвращает указатель (handle) на открытый файл, для дальнейшей работы с ним. Параметры: **sFile** - имя файла, **fmMode** - режим, в котором должен быть открыт файл (чтение/запись), **fsShare** - определяет, будет ли доступен файл другим пользователям/приложениям и как именно он будет доступен (чтение, запись и т.д.), **ftType** - формат текстового файла при открытии (доступен только в SilkTest International). Например:

**Code**

```
[ ] HFILE hFile = FileOpen ("c:\file.txt", FM_WRITE)
```

2. Открыть файл "c:\file.txt" для записи. При этом содержимое файла, если он уже существует, стирается. **hFile** в данном случае - переменная, с которой мы будем работать дальше, записывая данные в файл. Режим **FM_WRITE** задает тип открытия файла (в данном случае для записи). Существуют следующие типы режимов открытия файлов:
   - **FM_READ** - только чтение
   - **FM_WRITE** - только для записи
   - **FM_UPDATE** - для добавления данных в начало файла
   - **FM_APPEND** - для добавления данных в конец файла
3. **FileClose (hFile)** - закрывает ранее открытый файл hFile. Например, после выполнения следующего кода на диске будет создан пустой файл "c:\file.txt":

**Code**

```
[ ] HFILE hFile = FileOpen ("c:\file.txt", FM_WRITE)
[ ] FileClose (hFile)
```

4. **FileReadLine (hFile, sLine), FileWriteLine (hFile, sLine)** - соответственно считывает и записывает очередную строку из/в файл. Функция **FileReadLine** возвращает TRUE до тех пор, пока не будет достигнут конец файла. В следующем примере файл заполняется случайными строками, после чего содержимое файла считывается в список строк lsContents:

**Code**

```
        [ ] HFILE hF = FileOpen ("c:\file.txt", FM_WRITE)
        [ ] INTEGER i
        [-] for i = 1 to 10
                [ ] FileWriteLine (hF, RandStr ("X(10)"))
        [ ] FileClose (hF)
        [ ]
        [ ] LIST OF STRING lsContents
        [ ] STRING sLine
        [ ] hF = FileOpen ("c:\file.txt", FM_READ)
        [-] while (FileReadLine (hF, sLine))
                [ ] ListAppend (lsContents, sLine)
        [ ] FileClose (hF)
        [ ] ListPrint (lsContents)
```

5. Обратите внимание, что данный фрагмент кода показан лишь в качестве примера. Для считывания содержимого файла в список строк удобнее воспользоваться одной из функций **SYS_GetFileContents, ListRead** или **SYS_ListRead.**
6. **FileReadValue (hFile, aValue), FileWriteValue (hFile, aValue)** - соответственно считывает и записывает в файл данные любого типа. Ниже показан пример записи и считывания разных типов данных.

**Code**

```
[ ] // Объявление этого типа должно находиться Вне тесткейсов и функции
Main!!!
[-] type MyType is record
        [ ] STRING sStringField
        [ ] BOOLEAN bBooleanField
        [ ] INTEGER iIntField

        [ ] INTEGER  i1 = 54, i2
        [ ] REAL   r1 = 13.8, r2
        [ ] STRING   s1 = "some string", s2
        [-] MyType   m1, m2
                [ ] m1.sStringField = "Some text"
                [ ] m1.bBooleanField = TRUE
                [ ] m1.iIntField = 132
        [ ]
        [ ] HFILE hF = SYS_FileOpen ("c:\file.txt", FM_WRITE)
        [ ] FileWriteValue (hF, i1)
        [ ] FileWriteValue (hF, r1)
        [ ] FileWriteValue (hF, s1)
        [ ] FileWriteValue (hF, m1)
        [ ] FileClose (hF)
        [ ]
        [ ] hF = FileOpen ("c:\file.txt", FM_READ)
        [ ] FileReadValue (hF, i2)
        [ ] FileReadValue (hF, r2)
        [ ] FileReadValue (hF, s2)
        [ ] FileReadValue (hF, m2)
        [ ] FileClose (hF)
        [ ]
        [ ] Print (i2, r2, s2, m2)
```

7. Выводимый в конце результат будет таким:

**Code**

```
54 13.800000 some string {Some text, TRUE, 132}
```

8. **FileSetPointer (hFile, iNewValue [, SetMode] )** - устанавливает позицию чтения/записи в символах. **hFile** - переменная типа **HFILE**, указывающая, для какого файла устанавливается позиция; **iNewValue** - количество символов, на которое сдвигается позиция в файле. Значение этого параметра задается в зависимости от того, какое значение передается в третьем параметре: **SetMode**. **SetMode** может принимать три значения:
   - **FP_START** (значение по умолчанию) - при таком значении параметра SetMode позиция iNewValue будет установлена на iNewValue символов вперед относительно начала файла. Значение iNewValue должно быть положительным;
   - **FP_END** - позиция чтения/записи сдвигается на **iNewValue** символов назад, начиная от конца файла. Значение **iNewValue** должно быть отрицательным;/
   - 
   - **FP_RELATIVE** - позиция чтения/записи сдвигается относительно текущей позиции, может принимать как положительные, так и отрицательные значения.

Ниже приведен пример использования функции FileSetPointer:

**Code**

```
[ ] STRING s = "abcdefghijklmno"
[ ]
[ ] HFILE hF = FileOpen ("c:\file.txt", FM_WRITE)
[ ] FileWriteValue (hF, s)
[ ] FileClose (hF)
[ ]
[ ] hF = FileOpen ("c:\file.txt", FM_READ)
[ ] FileSetPointer (hF, 5, FP_START)
[ ] FileReadLine (hF, s)
[ ] Print (s)
[ ]
[ ] FileSetPointer (hF, -10, FP_END)
[ ] FileReadLine (hF, s)
[ ] Print (s)
[ ]
[ ] FileSetPointer (hF, -7, FP_RELATIVE)
[ ] FileReadLine (hF, s)
[ ] Print (s)
[ ]
[ ] FileClose (hF)
```

Результат работы:

**Code**

```
[ ] efghijklmno"
[ ] ijklmno"
[ ] lmno"
```

Кроме вышеперечисленных в SilkTest-е также есть функции **SYS_FileOpen, SYS_FileClose, SYS_FileReadLine, SYS_FileWriteLine, SYS_FileReadValue, SYS_FileWriteValue** и **SYS_FileSetPointer**. Их отличие от описанных функций в том, что исполняются они не процессом SilkTest-а, а процессом Агента. Это значит, что если во время тестирования необходимо подключаться к Агенту на удаленной машине и создавать файл на той самой машине, на которой установлен этот Агент, то необходдимо использовать функции **SYS_...**

Например, в следующем примере мы создадим файл file.txt в корневом каталоге диска C: на машине с именем MY_SERVER:

**Code**

```
[ ] HMACHINE hM = Connect ("MY_SERVER")
[ ] SetMachine (hM)
[ ]
[ ] HFILE hFile = SYS_FileOpen ("c:\file.txt", FM_WRITE)
[ ] SYS_FileClose (hFile)
```

Еще функции для работы с файлами:

- **SYS_FileExists** - проверяет, существует ли файл
- **SYS_CopyFile** - копирование файла. Если файл уже существует в папке, куда осуществляется копирование, сгенерируется исключение **E_SYSTEM:** "*** Error: File '...' already exists"
- **SYS_MoveFile** - перенос файла. Генерируется исключение если файл-источник не существует, либо файл-приемник уже существует
- **SYS_RemoveFile** - удаление файла. Файл не должен иметь аттрибутов "Только для чтения", "Скрытый" и "Системный"

Кроме того, в SilkTest-е существует группа функций для работы с **ini-файлами**, однако они рассмотрены ниже в главе 11.2.2 Работа с ini-файлами

### 11.1.2 Работа с каталогами

Для работы с каталогами в SilkTest-е предусмотрены следующие функции:

- **SYS_GetDir** - возвращает текущий (рабочий) каталог
- **SYS_SetDir** - устанавливает текущий (рабочий) каталог
- **SYS_MakeDir** - создать новый каталог (только одного уровня вложенности, т.е. если вы захотите создать каталог One, а в нем каталог Two, то вам придется вызвать эту функцию дважды)
- **SYS_RemoveDir** - удаляет каталог (только если он пустой, в противном случае из него необходимо удалить все файлы и вложенные каталоги)
- **SYS_DirExists** - проверяет, существует ли каталог; возвращает TRUE, если каталог существует
- **SYS_GetDirContents** - возвращает содержимое каталога. Возвращаемый тип - список **LIST OF FILEINFO**. Тип **FILEINFO** содержит в себе все параметры файла: имя, размер, аттрибуты и т.д.
- **SYS_GetExecutableDir** - возвращает местоположение файла **agent.exe**

- **SYS_GetDrive, SYS_SetDrive** - возвращают и устанавливают текущий диск соответственно

В качестве примера работы с каталогами приведем функцию, которая позволяет удалять каталоги вместе со всем их содержимым:

**Code**

```
[-] void RemoveDir (STRING sDir)
        [ ] LIST OF FILEINFO lfInfo
        [ ] INTEGER i
        [ ] STRING sPrevDir
        [-] if(SYS_DirExists(sDir))
                [ ] SYS_SetDir(sDir)
                [ ] lfInfo = SYS_GetDirContents(sDir)
                [-] for i = 1 to ListCount(lfInfo)
                        [-] if(lfInfo[i].bIsDir)
                                [ ] sPrevDir = SYS_GetDir()
                                [ ] RemoveDir (SYS_GetDir() + "\" +
lfInfo[i].sName)
                                [ ] SYS_SetDir(sPrevDir)
                        [-] else
                                [ ] SYS_RemoveFile(lfInfo[i].sName)
                [ ] SYS_SetDir (GetExecutableDir ())
                [ ] SYS_RemoveDir (sDir)
```

В приведенной функции есть одна особенность: если какой-либо из файлов имеет аттрибут "Только для чтения", "Скрытый" или "Системный", то сгенерируется исключение "***** Error: Permission denied to '...'**" для этого файла. Для того, чтобы все же удалять и такие файлы, необходимо воспользоваться **WinAPI** функцией **SetFileAttributesA** из файла **kernel32.dll** прежде, чем удалять файл, в самом начале цикла **for** (подробнее о DLL и WinAPI смотри главу 11.6 Использование Windows API и DLL)

## 11.2 Работа с реестром и ini-файлами

SilkTest имеет широкие возможности для работы с реестром и **ini-файлами.** И хотя в справке по SilkTest-у работа с ini-файлами рассматривается в разделе "Работа с файлами", мы решили совместить их описание в одном разделе с описанием работы с реестром, так как и то, и другое используется приложениями с одной целью: для сохранения данных в более-менее универсальном виде.

### 11.2.1 Работа с реестром

Функции для работы с реестром определены в файле **Registry.inc**, который находится в папке, где установлен SilkTest (обычно *"C:\Program Files\Segue\SilkTest"*). Однако по умолчанию этот файл не подключен как **Use File.** Если вам необходимо работать с реестром необходимо, прежде всего, подключить этот файл в диалоговом окне **Runtime Options**, либо с помощью деректривы **use.**

Ниже перечислены функции для работы с реестром. Первый параметр для всех этих функций - это целочисленное значение, соответствующее определенному разделу реестра (**HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE** и т.д.). Константы с соответствующими именами определены в файле **mswconst.inc.**

- **Reg_CreateKey (hRootKey, sKeyPath)** - создает новый раздел (Key) вместе с подразделами (если не существовали)
- **Reg_DeleteKey (hRootKey, sPath)** - удаляет существующий раздел
- **Reg_DeleteKeysAll (hRootKey, sPath)** - удаляет существующий раздел вместе с подразделами и значениями (Value)
- **Reg_CreateValue (hRootKey, sPath, sValueName, sValueData)** - создает новое значение sValueName в разделе sPath
- **Reg_DeleteValue (hRootKey, sPath, sValueName)** - удаляет существующее значение sValueName
- **Reg_EnumKeys (hRootKey, sPath)** - возвращает список всех подразделов в указанном разделе
- **Reg_EnumKeysAll (hRootKey, sPath)** - возвращает список всех подразделов в указанном разделе и во всех имеющихся в нем подразделов
- **Reg_EnumValues (hRootKey, sPath)** - возвращает список всех значений в указанном разделе
- **Reg_EnumValuesAll (hRootKey, sPath)** - озвращает список всех значений в указанном разделе и во всех имеющихся в нем подразделов
- **SYS_GetRegistryValue (iKey, sPath, sItem [, bConvert])** - возвращает величину указанного значения
- **SYS_SetRegistryValue (iKey, sPath, sItem, sValue)** - устанавливает значение

Ниже показан пример, в котором создается новый раздел в корневом разделе **HKEY_CURRENT_USER**, затем в нем создаются новые значения, выводятся на печать, после чего все созданное скриптом удаляется из реестра.

**Code**

```
        [ ] Print ("* * * Creating Keys and Values")
        [ ] Reg_CreateKey (HKEY_CURRENT_USER,
"Test_Key\Test_SubKey1\Test_SubKey2")
        [ ] Reg_CreateValue (HKEY_CURRENT_USER,
"Test_Key\Test_SubKey1\Test_SubKey2", "StringValue", "Test_String")
        [ ] Reg_CreateValue (HKEY_CURRENT_USER,
"Test_Key\Test_SubKey1\Test_SubKey2", "DWORDValue", "REG_DWORD: 0x1a")
        [ ] Reg_CreateValue (HKEY_CURRENT_USER, "Test_Key\Test_SubKey1",
"BinaryValue", "REG_BINARY: 01 13 7a")
        [ ]
        [ ] Print ("* * * Reading Data from Values in Key 'Test_SubKey2'")
        [ ] ListPrint (Reg_EnumValues (HKEY_CURRENT_USER,
"Test_Key\Test_SubKey1\Test_SubKey2"))
        [ ]
        [ ] Print ("* * * Reading Data from Values in Key 'Test_Key' and
below")
        [ ] ListPrint (Reg_EnumValuesAll (HKEY_CURRENT_USER, "Test_Key"))
        [ ]
        [ ] Print ("* * * Reading Data from Key 'Test_SubKey1' and below")
        [ ] ListPrint (Reg_EnumKeys (HKEY_CURRENT_USER,
"Test_Key\Test_SubKey1"))
        [ ]
        [ ] Print ("* * * Reading Data from Key 'Test_Key' and below")
        [ ] ListPrint (Reg_EnumKeysAll (HKEY_CURRENT_USER, "Test_Key"))
        [ ]
        [ ] Print ("* * * Reading Value 'DWORD'")
        [ ] Print (SYS_GetRegistryValue (HKEY_CURRENT_USER,
"Test_Key\Test_SubKey1\Test_SubKey2", "DWORDValue"))
        [-]
        [ ] Print ("* * * Deleting DWORD Value")
```

```
        [ ] Reg_DeleteValue (HKEY_CURRENT_USER,
"Test_Key\Test_SubKey1\Test_SubKey2", "DWORDValue")
        [ ]
        [ ] Print ("* * * Deleting Test_SubKey2 Key")
        [ ] Reg_DeleteKey (HKEY_CURRENT_USER,
"Test_Key\Test_SubKey1\Test_SubKey2")
        [ ]
        [ ] Print ("* * * Deleting Test_Key Key")
        [-] Reg_DeleteKeysAll (HKEY_CURRENT_USER, "Test_Key")
```

### 11.2.2 Работа с ini-файлами

Для работы с ini-файлами в SilkTest-е предусмотрены следующие функции:

- **IniFileOpen (sFile [, ftType])** - открывает файл для чтения и записи.Возвращает переменную типа HINIFILE
- **IniFileClose (hIniFile)** - закрывает ранее открытый ini-файл
- **IniFileGetValue (hIniFile, sSection, sName)** - возвращает строковое значение для заданной секции (sSection) и строки (sName)
- **IniFileSetValue (hIniFile, sSection, sName, sValue)** - устанавливает строковое значение sValue для заданной секции (sSection) и строки (sName)

В следующем примере создается новый файл c:\file.ini (или открывается существующий), после чего в секцию **new_section** для строки **new_value** записывается случайно сгенерированная строка, после чего она считывается и выводится в файл результата:

**Code**

```
        [ ] HINIFILE hIni = IniFileOpen ("c:\file.ini")
        [ ] IniFileSetValue (hIni, "new_section", "new_value", RandStr
("X(10)"))
        [ ] Print (IniFileGetValue (hIni, "new_section", "new_value"))
        [-] IniFileClose (hIni)
```

## 11.3 Использование проектов

В SilkTest-е существует понятие "Проект". В проект можно добавлять любые файлы, которые относятся к текущему проекту и могут понадобиться во время работы. Удобство проета заключается в том, что открыть нужный файл проще из списка файлов в окне проекта, чем искать его на диске, используя диалог открытия файла (File - Open). Кроме того, гораздо быстрее осуществляется доступ к используемым в подключенных к проету файлах классам, функциям, константам, аппстейтам и т.д.

К недостаткам проекта можно отнести тот факт, что окно проекта всегда занимает часть экрана SilkTest-а, что уменьшает видимую часть текущего открытого файла. Кроме того, окно проекта может мешать людям, которые просто привыкли всегда открывать файлы через меню File - Open, т.к. для них это окно будет просто бесполезно.

Для создания нового проекта необходимо выбрать пункт меню File - New Project. После чего можно выбрать пункт **"AutoGenerate Project"**, либо **"Create Project"**. В первом случае SilkTest попросит указать файл, из которого будут браться ссылки на файлы для нового проекта (файлы тестплана по умолчанию), во втором случае необходимые файлы придется добавлять вручную. При создании нового проекта в папке, которую вы указываете в окне диалога **"Create Project"**, будет создана новая папка с именем,

соответствующим имени нового проекта, куда и будут помещены файлы проекта (**ini-файл** и **vtp-файл**). Для создания проекта **TestApp** мы использовали второй вариант.

Обратите внимание: при переносе файлов проекта на другую машину, или просто на другой диск/в другую папку, необходимо сделать изменения в файле ***.vtp**. В разделе **[ProjectProfile]** параметр **ProjectIni** должен быть полным путем к **INI-файлу**, который находится в той же папке, что и файл проекта. Относительный путь для этого параметра недопустим.

## 11.4 Сохранение и использование сохраненных настроек

В SilkTest-е существует возможность сохранить имеющиеся настройки для дальнейшего их использования (например, при запусках на других машинах). Эти настройки хранятся в ***.opt** файлах. По умолчанию настройки SilkTest-а хранятся в файле **partner.ini**, который находится в папке, где установлен SilkTest. Информация о подключенных расширениях хранится в двух местах: для **host machine** - в **partner.ini**, для **target machine** - в файле **extend.ini** в той же папке.

Для сохранения текущих настроек в отдельном *.opt файле необходимо выбрать пункт меню *Options - Save New Options Set*, после чего указать имя и местоположение файла. В итоге будет создан файл, в котором хранятся все настройки SilkTest-а. Для открытия ранее сохраненных настроек необходимо выбрать пункт меню *Options - Open Options Set* и выбрать необходимый файл.

Для того, чтобы вернуться к настройкам по умолчанию, необходимо закрыть текущий Options Set с помощью пункта меню *Options - Close Options Set*. При использовании расширений кроме подключения файла настроек **(*.opt)** также придется копировать в папку SilkTest-а требуемый файл **extend.ini**, чтобы подключить настройки для **target machine**.

Кроме того, можно подключать файл настроек во время запуска SilkTest-а из командной строки. Подробнее о параметрах командной строки см. раздел 11.5 Параметры командной строки SilkTest-а.

## 11.5 Параметры командной строки SilkTest-а

Иногда полезно запускать SilkTest из командной строки (например, для запуска тестирования в определенное время). Ниже перечислены параметры командной строки файла **partner.exe**, который является запускаемым файлом для SilkTest-а.

Общий формат командной строки выглядит так:

**Code**

```
partner.exe [опции] имя_файла [аргументы]
```

Опции могут быть следующими:

- **-complog <filename.txt>** - при указании этой опции SilkTest будет писать в указанный файл filename.txt ошибки компиляции, которые возникают во время работы скрипта/тестплана
- **-m <target_machine_name>** - указывает target machine. на которой будет произведен запуск. По умолчанию используется local. Для того, чтобы эта опция сработала, необходимо, чтобы в настройках SilkTest-a (Options - Runtime - Network) был включен один из протоколов (**TCP/IP** или **NetBIOS**), или чтобы данная опция была прописана в подключаемом ***.opt** файле (см. ниже)
- **-opt <opt_file_name>** - позволяет производить запуск с подключенным файлом опций opt_file_name
- **-p** - используется в случае, если SilkTest вызывается из другой программы в режиме "batch job", позволяя ей получать сообщения о количестве ошибок, произошедших в скрипте
- **-proj <project_name>** - позволяет открыть erfpfyysq проект project_name при запуске SilkTest-a
- **-q** - закрыть SilkTest после завершения работы скрипта/тестплана
- **-query <query_name>** - указывает запрос query_name, который должен выполниться при запуске тестплана
- **-r <testplan/script_name>** - эта опция должна указываться последней в списке опций. Непосредственно после нее должно быть имя скрипта/тестплана, который необходимо запустить
- **-resexport** - автоматически экспортировать суммарную информацию по прошедшим тесткейсам в ***.rex** файл. Указание этой опции аналогично действию функции **ResExportOnClose**
- **-resextract** - автоматически экспортировать суммарную информацию по прошедшим тесткейсам в ***.txt** файл.
- **[аргументы]** - в этой части командной строки можно перечислить аргументы, передаваемые в скрипт. Внутри тесткейсов эти аргументы можно получить с помощью функции **GetArgs()**, которая возвращает список переданных строк. Если параметр состоит из более чем одного слова, необходимо заключить его в двойные кавычки.

Ниже показан пример, который запускает SilkTest, подключает ***.opt** файл, подключается к компьютеру RUNSERVER через порт 1111, запускает скрипт notepad.t с одним параметром "Test Args", после чего закрывает SilkTest. Ошибки компиляции пишутся в файл **errors.txt**, суммарная информация выводится в файл **notepad.txt**.

**Code**

```
partner.exe -q -opt c:\options.opt -complog c:\errors.txt -m RUNSERVER:1111 -resextract -r C:\notepad.t "Test agrs"
```

В SilkTest-е начиная с версии 8.0 появились еще два параметра командной строки:

- **-smlog <file_name>** - позволяет сохранять сообщения SilkMeter-a в указанном файле
- **-quiet** - запускает SilkTest в "тихом режиме", при котором блокируются всплывающие окна SilkMeter-a. Этот параметр рекомендуется использовать в том случае, если тестирование происходит в полностью автоматическом режиме, когда пользователь не имеет возможности отреагировать на появляющиеся сообщения

SilkMeter-a. При использовании этого параметра также рекомендуется использовать предыдущий параметр **smlog.**

## 11.6 Использование Windows API и DLL

Возможности SilkTest-a широки, однако при тестировании практически всегда возникают задачи, которые невозможно решить встроенными средствами. Для таких случаев SilkTest предоставляет возможность использовать функции Windows API, а также любые функции из DLL-файлов.

Общий вид объявления функции выглядит так:

**Code**

```
[+] dll dllname.dll
        [ ] [return-type] func-name ( [arg-list] ) [alias dll-fname]
```

где: **dllname.dll** - собственно имя dll файла, функции которого предполагается использовать **return-type** - возвращаемый тип. Необязательный параметр **func-name** - имя функции. Может отличаться от имени, которое указано в DLL. Если имя функции указывается другое, то необходимо указать последний параметр alias **arg-list** - список принимаемых аргументов. Необязательный параметр **alias dll-fname** - имя функции, как оно указано в DLL. Указывается только в том случае, если имя **func-name** отличается от оригинального имени функции.

Простейший пример функции, которая блокирует рабочую станцию:

**Code**

```
[-] dll "user32.dll"
        [ ] long LockComputer () alias "LockWorkStation"
[ ]
[-] main ()
        [ ] LockComputer ()
```

При запуске данной функции main компьютер будет заблокирован. Мы использовали новое имя **LockComputer** лишь для демонстрации того, как это сделать. Некоторые из функций уже описаны в файле **mswfun32.inc**. Для того, чтобы получить к ним доступ, необходимо подключить к **Use** файлам файл **msw32.inc**, который подключает все необходимые файлы.

В качестве еще одного примера приведем функцию для извлечения цвета пикселя. Для этого нам потребуются две API функции: первая из них извлекает ссылку на контекст устройства (Handle to a display device context, **HDC**), используя **Window ID**, а вторая извлекает собственно цвет пикселя по известному **HDC**. Для того, чтобы каждый раз затем не вызывать обе этих функции по очереди, мы напишем простую функцию **GetPixelColor**, которая будет принимать в качестве аргументов **Window ID** и координаты пикселя, цвет которого необходимо узнать.

**Code**

```
[-] dll "user32.dll"
        [ ] LONG GetDC(HWND hWnd)
[-] dll "gdi32.dll"
        [ ] LONG GetPixel(LONG hdc,INT x,INT y)
[ ]
[-] LONG GetPixelColor (HWND hWnd, INT x, INT y)
        [ ] return GetPixel (GetDC (hWnd), x, y)
```

# 12. Справочная информация

## 12.1 Типы файлов SilkTest

**SilkTest** использует много разных типов файлов: файлы проекта, фалы фреймов, файлы тестпланов и т.д. В этой главе перечислены все типы файлов, с которыми пользователь может столкнуться, и их описания.

В частности, если вы работаете в команде, то скорее всего используете какую-то систему контроля версий (MKS, SVN, TFS и т.п.) для поддержания целостности проекта. Не все файлы, которые использует **SilkTest,** необходимо подкладывать под систему контроля версий. Например, файлы скомпилированного кода и временные файлы держать в системе контроля версий нет смысла, так как во-первых, пользователь не имеет никакого отношения к изменениям в этих файлах, а во-вторых, они могут слишком часто меняться.

Ниже приведена таблица расширений, которые используются для файлов **SilkTest** с описанием их назначения. * Звездочкой отмечены файлы, которые не рекомендуется хранить в системе контроля версий.

| Расширение | Назначение файла |
|---|---|
| .vtp | **Файл проекта.** Файл имеет формат обычного INI-файла и содержит информацию об именах и местоположении файлов проекта. Обратите внимание, что относительные пути в vtp файле недопустимы. Каждому vtp файлу соответствует INI-файл (projectname.ini) с настройками проекта |
| .pln | **Файл тестплана** |
| .inc | **Фрейм-файл.** В этих файлах обычно хранятся описания винклассов, окон, констант, функций и т.п. |
| .t | **Файл тесткейсов.** В этих файлах обычно хранятся записанные и написанные вручную тесткейсы |
| .g.t | **Файл тесткейсов, управляемых данными.** В случае, если обычный тесткейс преобразуется в тесткейс, управляемый данными, средствами SilkTest-а, то он будет сохранен со старым именем и новым расширением .g.t |
| .res | **Файл результатов.** Здесь хранятся результаты запусков тесткейсов. Этот файл не отмечен звездочкой, однако помещать его в систему контроля версий следует только в том случае, если не используется самописный лог и результаты работы тесткейсов определяются с использованием встроенных возможностей лога **SilkTest** |

| | |
|---|---|
| .s | **Файлы тесткомплексов (Test Suite).** Если у вас имеется несколько файлов с тесткейсами (.t), вы можете перечислить их все в файле тесткомплекса и запускать их все из одного места. |
| * .to, .ino | **Объектные (скомпилированные) файлы** тесткейсов и фрейм-файлов соответственно |
| * .in_, .t_ | **Временные файлы**, хранящие предыдущую копию модифицированного файла |

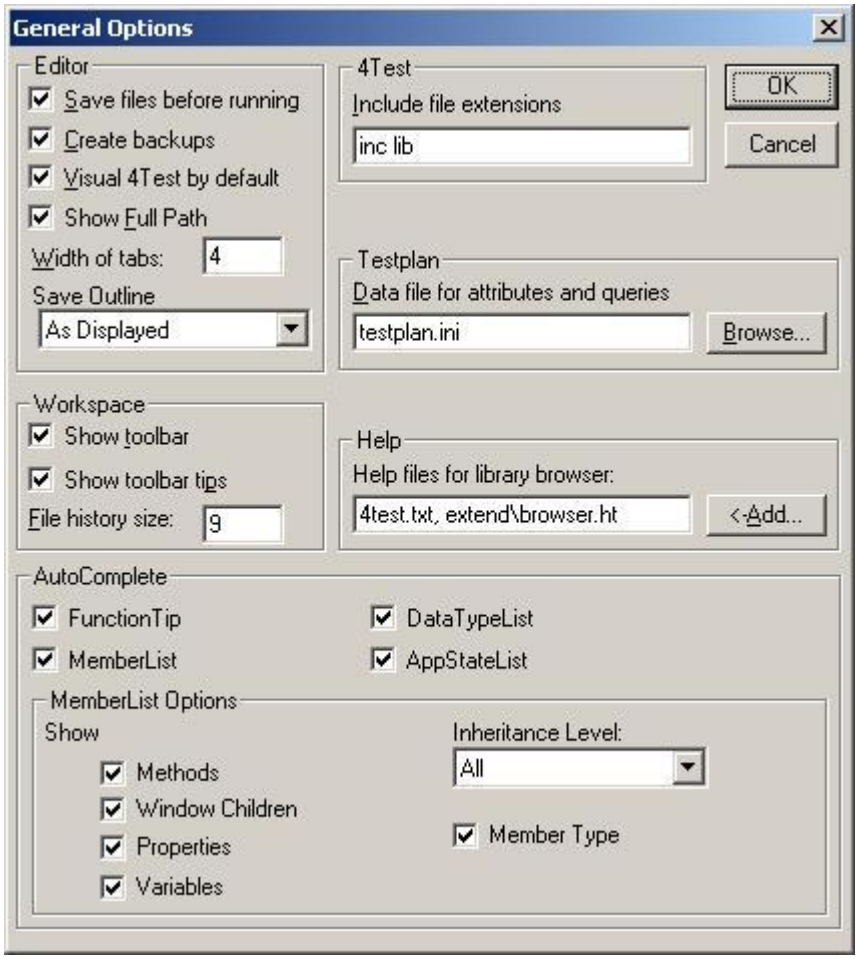
Отдельного внимания заслуживают объектные файлы, так как с ними связаны два весьма важных обстоятельства.

1. **Средство сокрытия кода.** При запуске и компиляции скриптов **SilkTest** прежде всего пытается открыть объектный файл. Если такого файла нет, **SilkTest** его создает. Следовательно, если у вас есть файл .t, в котором подключается внешний файл .inc (с помощью инструкции use), совсем необязательно хранить сам .inc файл, достаточно хранить его скомпилированный вариант .ino. Это может быть полезно, если вам по какой-то причине необходимо скрыть реализацию кода от заказчика. В таком случае достаточно отдать ему .t файл и подключаемый .ino файл, и никто не сможет прочитать исходный код .inc файла. Однако надо быть очень осторожным и быть на 100% уверенным, что скомпилированный код без проблем отработает на другом компьютере
2. **Проблема компиляции.** Иногда после внесения изменений остается ощущение, что на самом деле изменения не вступили в силу. Такое действительно иногда случается: по какой-то причине **SilkTest** "не видит" сделанных изменений и не перекомпилирует соответствующие файлы. В таких случаях необходимо удалить объектные файлы и повторить компиляцию.

## 12.2 Полезные настройки SilkTest

В этой главе мы хотим рассказать о наиболее полезных настройках, которые могут использоваться в работе. Мы не будем говорить обо всех настройках, так как некоторые из них настолько просты и понятны, что читатели без труда разберутся в них сами. Однако некоторые параметры могут существенно повлиять на работу SilkTest и поэтому их необходимо описать подробно.

### General Option (меню Options - General)



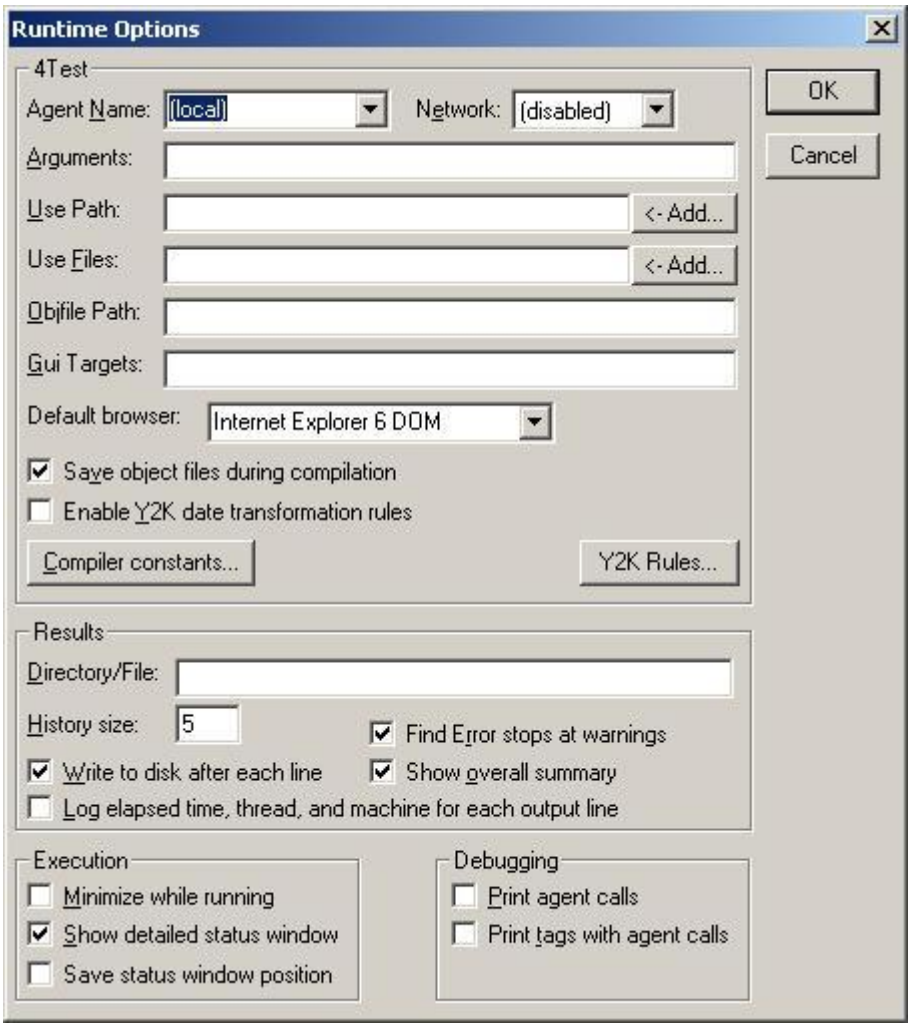

К наиболее важным настройкам здесь относятся:

- **Save files before running.** Желательно чтобы эта опция была включена, тогда SilkTest автоматически сохраняет все изменения перед запуском скриптов. Иначе, если SilkTest зависнет и его придется закрывать через Task Manager, последние изменения не сохраняться
- **Create backups.** Данная опция устанавливает, будут ли создаваться резервные копии файлов во время их редактирования (файлы с расширениями **.t_** и **.in_**)
- **Save Outline.** Данная опция регулирует, как будут сохраняться блоки кода при сохранении файлов. *As Displayed* – как они выглядят на экране (используйте этот пункт если вы часто работаете с одними и теми же файлами); *Collapse All* – все блоки будут свернуты независимо от того, как они выглядят при редактировании (эта опция полезна если вы не очень часто редактируете файлы и позволяет не задумываться в таких случаях, какие блоки были развернуты, а какие нет); *Expand All* – все блоки при сохранении будут развернуты. Здесь следует иметь ввиду, что открытие/закрытие какого-либо блока кода в редакторе SilkTest влияет непосредственно на файл (если открыть файл SilkTest в любом другом редакторе, то можно увидеть, что закрытые блоки отмечены символами **[+]**, а открытые – символом **[-]**). Поэтому если вы храните скрипты в системе контроля версий, вам необходимо придерживаться каких-то строгих правил (например, при осуществлении операции check-in все блоки должны быть закрыты). Иначе система будет думать, что в файл были внесены изменения, хотя на самом деле изменений

никаких не было. Кроме того, обратите внимание на пункты меню *Outline - Collapse All/Expand All*, они также могут быть полезны

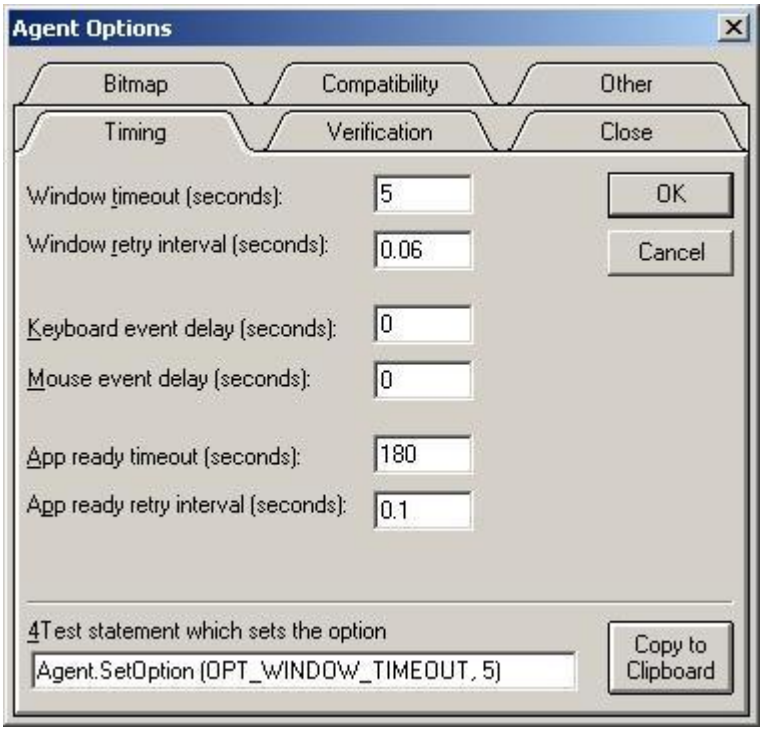
- **Autocomplete.** Здесь можно настраивать окно подсказок, которое всплывает при редактировании скриптов. *FunctionTip* – задает, показывать ли подсказку для функций; *MemberList* – показывать ли список автозаполнения для методов винклассов; *DataTypeList* – автозаполнение для различных типов данных (например, *LIST OF...*); *AppstateList* – показывать список доступных аппстейтов. Обычно удобнее всего включить все эти опции.
- **MemberListOption.** Если включена опция *MemberList*, то можно настраивать, как именно будет показан список автозаполнения. *Show Methods/Window Children/Properties/Variables* – устанавливает, какие элементы винкласса показывать в списке автозаполнения. Рекомендуется включать все опции
- **Inheritance level.** Устанавливает "глубину" вложенности, на которую будет произведен поиск свойств, методов и т.п. При значении *None* будут показаны только члены текущего объекта; при значении *Below AnyWin class* – только члены винклассов, которые находятся "ниже" объекта *AnyWin* (например, методы *Exists* и *Click* видны не будут, так как они определены для класса *AnyWin*); *All* – будут отображены все элементы, включая члены класса *AnyWin*. Рекомендуется установить здесь значение *All*, а также включить опцию *Member Type*, тогда рядом с именами переменных будет указываться их тип

### Runtime Options (меню Options - Runtime)



- **Agent Name/Network.** Здесь указывается, какой агент будет выполнять скрипты и какой протокол используется для передачи данных. По умолчанию используется локальный агент (local)
- **Use Path.** Здесь можно указать один или несколько (разделенных запятой) путей, по которым SilkTest будет искать .inc файлы, которые подключаются с помощью директивы use в скриптах
- **Use Files.** Здесь можно подключить дополнительные **.inc** файлы, которые используются в ваших скриптах. Зачастуюименно эта опция служит причиной того, что проект не компилируется, если был добавлен какой-то файл, а затем удален за ненадобностью (SilkTest автоматически добавляет сюда имя файла при создании фрейма из меню *File - New - Test Frame*)
- **Compiler constants.** При нажатии на эту кнопку открывается окно, в котором можно задать глобальные константы, которые будут видны во всех скриптах
- **Секции Results и Debugging.** Здесь производятся настройки результатов. Они подробно описаны в главе 7. Обработка результатов (Results)
- **Секция Execution.** *Minimize while running* позволяет сворачивать SilkTest во время выполнения скриптов; две другие опции управляют поведением окна статуса выполнения (показывать/не показывать и запоминать позицию).

### Agent Option (меню Options - Agent Options)




Все настройки в этом окне управляют поведением Агента и могут быть изменены динамически во время работы скриптов (см. главу 10.3 Динамическое изменение настроек Агента). При изменении различных настроек в поле ввода внизу окна отображается код, с помощью которого можно изменить эту опцию динамически.

- **Вкладка Timing.** На этой вкладке устанавливаются различные временны*е* настройки: время ожидания окна (*Window timeout*) и интервал задержек между попытками активировать окно (*Window retry interval*); задержки при вводе текста (Keyboard event delay) и передвижения мыши (*Mouse event delay*); время ожидания готовности приложения (*App ready timeout*) и интервал задержки между опросом

приложения на готовность (*App ready retry interval*). Первые две настройки полезны для настройки ожидания окон в методах типа *SetActive* и *Exists*. Реакция SilkTest на появление окна также зависит от первых четырех настроек вкладки **Verification** (см. ниже). Две вторых опции полезны для замедления выполнения скриптов (например, в отладочных целях или чтобы визуально оценить работу скрипта). Последние две управляют временем ожидания полной загрузки приложения

- **Вкладка Verification.** Первые 4 опции заставляют SilkTest проверять, что объекты соответственно активны, разблокированы (не "задисейблены"), доступны (отвечают на запросы системы) и уникальны (т.е. что нет двух объектов, которые могут соответствовать одному тегу). Первые три из этих опций рекомендуется отключать, так как SilkTest часто выдает ошибку, хотя может работать с объектами. *Verify that coordinates passed to a method are inside the window* – проверяет, что координаты для нажатия кнопки мыши не выходят за границы объекта, прежде чем осуществить собственно нажатие кнопки. *Verify that an application is ready* – заставляет Агент синхронизироваться с тестируемым приложением, ожидая пока оно станет доступным (требуется подключение расширения). На скриншоте показаны рекомендуемые настройки для этой вкладки.


- **Вкладка Close.** Здесь перечислены заголовки кнопок, пункты меню и сочетания клавиш, которые используются по умолчанию для закрытия окон. Если окна в вашем приложении закрываются нестандартным способом, то можно добавить этот способ здесь. Например, если диалоговые окна в приложении закрываются с помощью клавиши *F12*, то в поле *Keystrokes used to close a dialog box window* необходимо ввести значение *""* (без кавычек)
- **Вкладка Bitmap.** Здесь задаются параметры проверки графических изображений. *Bitmap match count* – задает количество проверок, которое должна пройти проверка графического изображения прежде, чем проверка будет считаться упсешной. Обычно достаточно одной проверки; *Bitmap match interval* – задает интервал, с которым будут осуществляться проверки изображений; *Bitmap match timeout* – задает интервал времени, в течение которого SilkTest ожидает стабилизации изображения; *Bitmap compare tolerance* – это очень важная опция при проверке графики, она задает количество пикселей, которые могут не совпадать при проверке, однако изображения все равно будут считаться одинаковыми

- **Вкладка Compatibility.** Эти настройки служат для обеспечения совместимости с предыдущими версиями SilkTest. Здесь, например, можно указать, что элемент управления radio list надо распознавать как отдельные кнопки, а не как один элемент управления, использовать старый тип тегов и т.п.
- **Вкладка Other.** Здесь перечислены настройки, которые не попали ни в одну из уже перечисленных категорий. Первые две опции (*Sizing/Moving tolerance*) используются для задания погрешности при изменении размеров и перемещении окон (некоторые окна невозможно передвинуть на точное количество пикселей или изменить точно их размер). Чем больше значения в этих полях – тем больше может быть погрешность. *Pick menus before getting menu item information* – заставляет SilkTest раскрывать меню прежде чем проверять, что пункты меню существуют либо включены. *Pick dropdowns before getting item information* – эта опция нужна только в тех случаях, если элементы выпадающего списка становятся видны лишь после того, как список раскрыт. В этом случае SilkTest будет раскрывать выпадающие списки прежде чем осуществлять какие-либо действия с ними. Еще две интересные опции *Show windows which are out of view* и Automatically scroll window into view. Первая из них заставляет Агента "видеть" элементы управления, которые не видны на экране, вторая заставляет SilkTest прокручивать элементы управления, если они находятся вне зоны видимости. Эти настройки особенно полезны в случае тестирования веб-приложений, где часто используются страницы со скроллерами и не все элементы одновременно видны на экране

## 12.3 Полезные клавиатурные сочетания

В этой главе перечислены наиболее часто используемые клавиатурные сочетания, которые могут оказаться полезными в работе.

| Сочетание клавиш | Описание |
|---|---|
| Alt+F9 | Компилировать проект |
| F9 | Запустить скрипт на выполнение |
| Shift+Shift | Останавить выполнение скрипта. Обратите внимание, что остановка не всегда происходит мгновенно. В некоторых случаях SilkTest дожидается окончания операции (например, отклик системы на какое-то действие). Особенно часто такое происходит в клиент-серверных приложениях |
| F12 | Это "магическая" клавиша, которая позволяет вам перейти к объявлению функции, метода или типа, на котором в данный момент находится курсор редактора. Не всегда срабатывает, если объявление находится в том же файле, где вы в данный момент находитесь |
| Alt+стрелки (←, ↑, →, ↓) | Передвигает выделенную часть кода в направлении нажатой стрелки (подробнее см. главу Несколько слов о редакторе SilkTest) |
| Alt+l+m | Закомментировать выделенный текст |
| Alt+l+n | Раскомментировать выделенный текст |
| Ctrl+цифровой плюс | Раскрыть блок |
| Ctrl+цифровой минус | Свернуть блок |

| | |
|---|---|
| Ctrl+F9 | Включить режим отладки |
| F5 (в режиме отладки) | Установить/снять точку прерывания (breakpoint) |
| F8 (в режиме отладки) | Выполнить одну строку (сделать "шаг") |
| F7 (в режиме отладки) | Зайти внутрь функции, находящейся на текущей строке |
| Alt+d+e (в режиме отладки) | Начать отладку сначала |
| Alt+d+x (в режиме отладки) | Выйти из режима отладки |