

Министерство образования Республики Беларусь  
Учреждение образования «Полоцкий государственный университет»

Факультет информационных технологий  
Кафедра технологий программирования

**Методические указания**  
к выполнению лабораторной работы №14

по дисциплине «**Надёжность программного обеспечения**»  
для специальности 1-40 01 01 Программное обеспечение информационных  
технологий

на тему «**Автоматизация тестирования для standalone-приложений**»

Новополоцк, 2017 г.

**Название:** «Автоматизация тестирования для standalone-приложений».

**Цель работы:** ознакомиться с автоматизированным тестированием standalone-приложений. Научиться составлять тесты с помощью Silktest.

## **Теоретическая часть**

### ***1 Работа в среде SilkTest 8.0***

#### ***1.1 Начало работы***

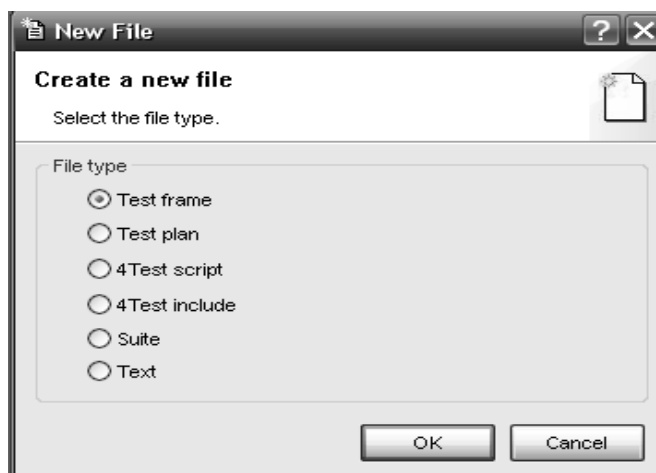
##### ***1.1.1 Основные понятия***

Основные понятия SilkTest: фрейм и тест-кейс. Во фрейме хранится информация об окнах, с которыми работает скрипт (например, кнопки, меню, списки). В тест-кейсе хранятся записанные или написанные действия, которые необходимо произвести во время тестирования. Информация об окнах обычно хранится в файлах с расширением \*.inc, тест-кейсы - в файлах с расширением \*.t.

#### **Создание фреймов**

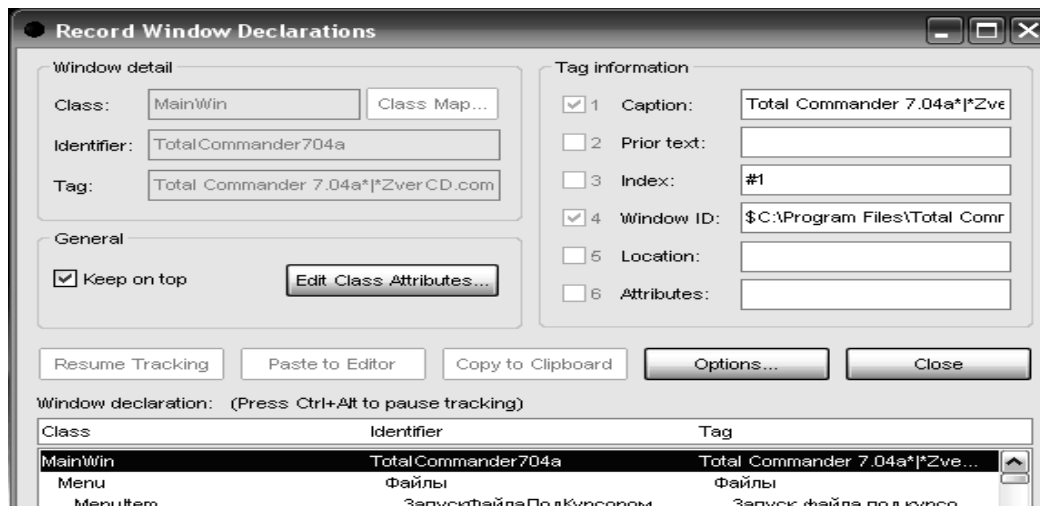
Существует два способа записи нового окна:

1 Меню File-New-Test Frame (рисунок 1).



**Рисунок 1** – Создание нового файла для записи

2 Меню Record-Window Declarations (рисунок 2).



**Рисунок 2 – Меню записи нового окна**

В первом случае необходимо выбрать открытое приложение, для которого создается фрейм, и SilkTest создаст объявление нового окна.

Во втором случае открывается дополнительное окно Record Window Declarations, в котором отображаются все объекты окна приложения, на котором в данный момент находится курсор мыши. Чтобы остановить процесс сканирования и поместить информацию о текущем окне в документ SilkTest, необходимо нажать комбинацию клавиш <Ctrl+Alt> и кнопку «Paste to Editor». В результате в текущий документ SilkTest добавится описание этого окна.

### **Структура тест-кейсов**

Инструкции тест-кейсов пишутся в файлах скриптов.

Файл скрипта с расширением \*.t. имеет следующую структуру:

- заголовок. Необязательный комментарий, характеризующий содержимое данного файла;
- блок подключения файлов. Набор Use-директив, которые подключают необходимые для данного скрипта файлы (как правило, это фреймы или файлы типа \*.inc);
- блок декларации внешних переменных. Область для объявления переменных, констант, которые используются различными тест-кейсами;
- тест-кейсы. Модули со сценариями автоматического тестирования. Каждый сценарий помещается в модули с ключевым словом testcase;
- функция main. Точка входа в программу.

### **Реализация технологии Record/PlayBack в среде SilkTest**

Реализация технологии Record/PlayBack в SilkTest рассмотрена для создания библиотек объектов и тестовых скриптов, для среды запуска скриптов и средств опознавания объектов.

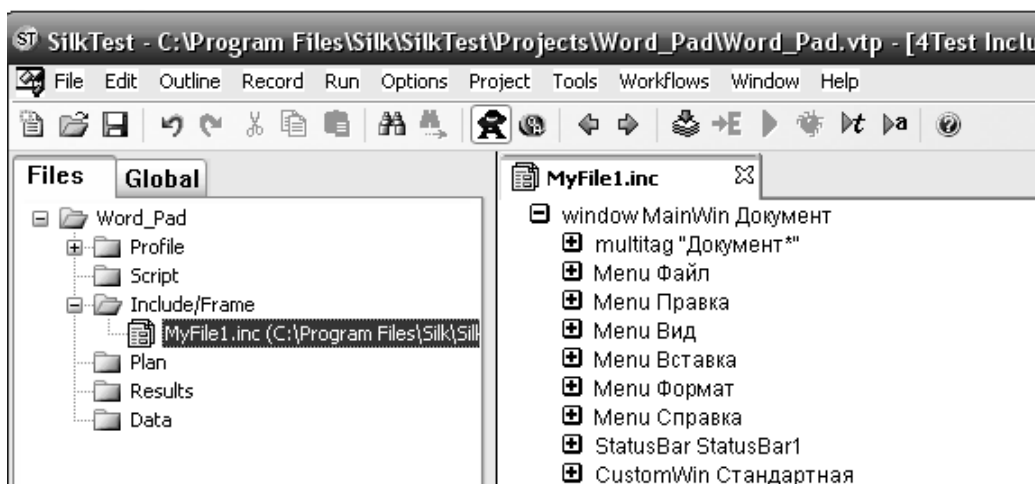
Создание библиотеки объектов

Для создания библиотеки объектов следует:

- 1 Запустить SilkTest.
- 2 Открыть тест-приложение, например «WordPad» .

- 3 Выбрать в меню SilkTest команду File-New-4Test Include file.
- 4 Сохранить его, например как MyFile1.inc.
- 5 Выбрать в меню SilkTest команду Record-Window Declarations.
- 6 Навести курсор мыши на тест-приложение.
- 7 Нажать комбинацию клавиш «Ctrl+Alt».
- 8 Нажать кнопку «Paste to Editor» на форме Record Window Declarations.

Рабочее окно SilkTest будет выглядеть, как представлено на рисунке 3.



**Рисунок 3 – Рабочее окно SilkTest**

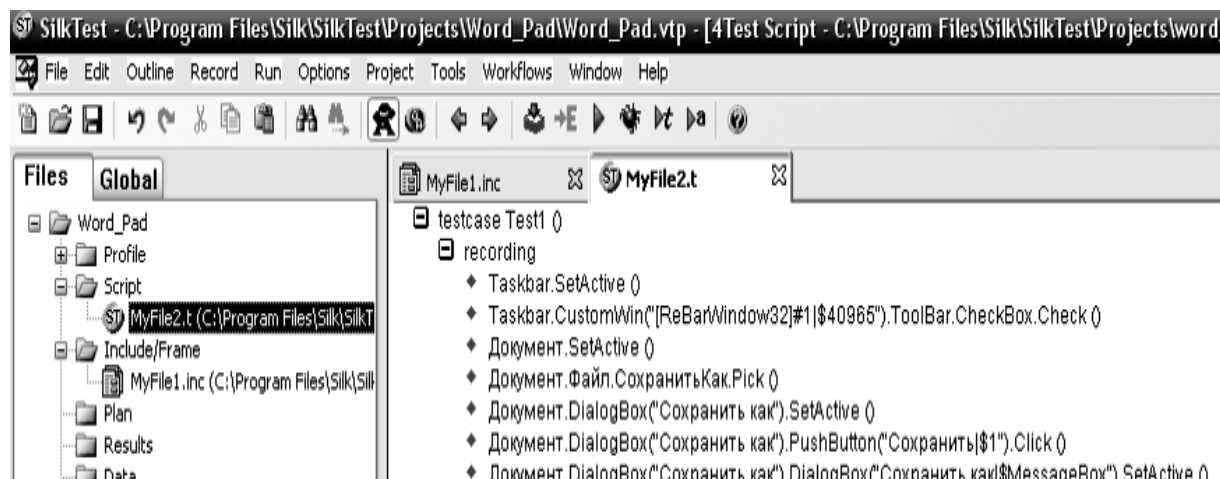
Для использования объектов библиотеки, в файл скрипта необходимо включить следующий код: *Use «MyFile1.inc»*. Этот файл можно открыть в любом текстовом редакторе.

### **Создание тестового скрипта**

Для создания тестового скрипта следует:

- 1 Запустить SilkTest.
- 2 Открыть тест-приложение, например «WordPad».
- 3 Выбрать в меню SilkTest команду File-New-4Test script.
- 4 Сохранить его, например как MyFile2.t.
- 5 Выбрать в меню SilkTest команду Record-Testcase.
- 6 Нажать кнопку «Start Recording».
- 7 Выполнить необходимые действия над тест-приложением.
- 8 Нажать кнопку «Done».
- 9 Нажать кнопку «Paste to Editor».

После выполнения выше представленного скрипта рабочее окно SilkTest будет выглядеть, как на рисунке 4.




**Рисунок 4 – Рабочее окно с примером скрипта**

Скрипт, созданный в режиме Recording, можно изменить и дополнить ручным способом.

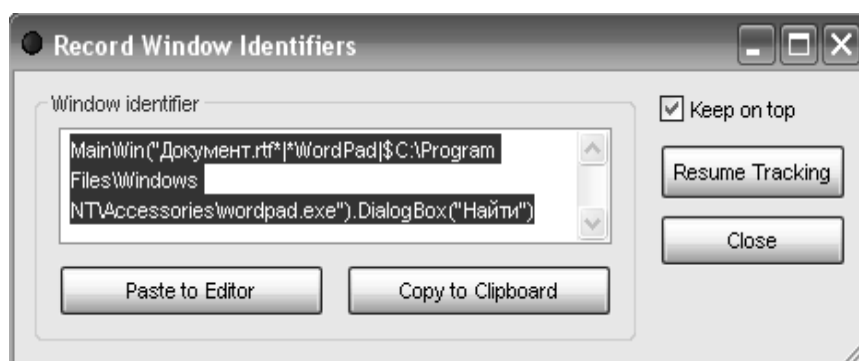
### **Среда запуска скриптов**

На рисунке 4 представлен фрагмент программного кода, выражающего предпринятые действия. Это готовый для исполнения код.

Все выполняемые строки кода находятся внутри блока testcase. Один скрипт может иметь множество тест-кейсов, исполняемых по очереди. Для того, чтобы выполнить лишь один тест-кейс, необходимо нажать кнопку  и выбрать нужное значение из списка доступных тест-кейсов.

### **Средство опознавания объектов**

Когда необходимо просмотреть описание конкретного объекта, следует использовать средство опознавания объектов SilkTest. Его можно вызвать следующим образом: Record-Window Identifiers. Чтобы получить информацию об объекте, необходимо навести на него курсор и нажать <Ctrl+Alt>. Описание можно скопировать в файл, нажав кнопку «Paste to Editor» (рисунок 5).



**Рисунок 5 – Средство опознавания объектов**

## ***1.2 Recovery-система***

При разработке тест-кейсов часто возникает необходимость написания программного кода для подготовки тестовой среды к запуску скриптов или выполнения некоторых предварительных действий, связанных с установкой определенных настроек SilkTest для конфигурации конкретной

машины или инициализации некоторых глобальных переменных. В качестве примера можно привести инициализацию величины, означающей время в секундах, в течение которого ожидается появление некоторого окна приложения. В разных средах тестируемое приложение работает с разными скоростями и соответственно на ожидание появления окна требуется разное время.

Инициализацию таких данных перед запуском скриптов можно осуществлять разными способами. Одним из способов является разработка функции или тест-кейса, который осуществит необходимую инициализацию и будет запускаться перед запуском других тест-кейсов. Разработку отдельной функции инициализации стартовых установок тестового приложения называют функцией AppState (Application State). Но поскольку каждый тест-кейс может запускаться как в группе (например, тестпланом), так и поодиночке, то инициализацию данных придется интегрировать в каждый тест-кейс. Вторым способом стартовой инициализации в SilkTest является использование специализированных функций Recovery-системы:

- 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.

Этот набор функций, который можно переопределить. В противном случае будут задействованы функции по умолчанию (DefaultScriptEnter, DefaultScriptExit, DefaultTestCaseEnter, DefaultTestCaseExit, DefaultTestPlanEnter, DefaultTestPlanExit).

Таким образом, Recovery-система – средство для выполнения некоторого программного кода до и после выполнения отдельного тест-кейса.

Особенностью Exit-функций Recovery-системы является то, что они принимают параметром BOOLEAN-значение, позволяющее установить, завершилась ли работа скрипта нормально или возникла исключительная ситуация.

### ***1.3 Создание тестплана (TestPlan)***

Тестплан в SilkTest – это последовательный набор инструкций для запуска тест-кейсов пакетом. Тестплан – это структурированный (часто иерархический) документ. Большие тестпланы могут быть поделены как master testplan и один или более subplans. Файлы, содержащие тестпланы имеют расширение \*.pln.

Для создания тестплана следует выбрать пункт меню File-New-Test Plan.

#### **Добавление тест-кейса в тестплан**

Для того, чтобы добавить в план тест-кейс, необходимо использовать нижеописанный синтаксис:

```
Script:scriptname.t,  
Testcase:testcasename,  
Testdata:testdata  
Optionset:filename.opt,
```

где Script, Testcase, Testdata, Optionset – ключевые слова;

scriptname.t – имя файла, содержащего тест-кейсы;

testcasename – название тест-кейса;

testdata – выражение вида: data[,data], где data – любое правомерное 4Test выражение;

filename.opt – подключаемый файл свойств.

Для соединения master plan с subplan необходимо добавить include выражение в master plan:

```
include:myinclude.pln,
```

где include – это ключевое слово, а myinclude.pln – файл плана, который является subplan.

Чтобы добавить комментарии к тест-кейсу следует использовать нижеописанный синтаксис:

```
comment:Yourcommenttext,
```

где comment – ключевое слово, а Yourcommenttext – комментарии в тестплане, которые будут отображаться в результатах выполнения тестов.

Пример кода для добавления в тестплан тест-кейса *CurrentAppStateTest* из скрипта *ScriptFile.t*:

```
Script: ScriptFile.t, //файл, содержащий тест-кейсы приложения  
Testcase: CurrentAppStateTest //тест, включаемый в тестплан  
Testdata: "01000101" //параметр для тест-кейса CurrentAppStateTest  
Comment: "Проверка текущих параметров приложения"
```

#### **Запуск тестплана (полный, выборочный)**

Для запуска тестплана на выполнение необходимо выбрать пункт меню Run-Run All Testcases или нажать <F9>. Существует возможность выборочного запуска тест-кейсов из тестплана. Для пометки тест-кейса нужно установить курсор на тест-кейсе и выбрать пункт меню Testplan-Mark. Для запуска помеченного множества тест-кейсов на выполнение необходимо выбрать пункт меню Run-Marked Tests. Вышеперечисленные действия можно выполнять, используя кнопки, расположенные под главным меню.

### **1.4 Использование внешних данных**

SilkTest позволяет помимо объявления переменных в тест-кейсах, использовать данные, которые считываются из какого-либо внешнего источника (базы данных, текстовых файлов определенного формата, электронной таблицы). Тест-кейсы, считывающие данные из внешнего источника, называются Data Driven Testcases (дословно: тест-кейсы,

управляемые данными). Такой тест-кейс представляет собой шаблон, используемый несколько раз, при этом каждый раз работа выполняется с новыми данными.

В каких случаях полезны эти возможности:

- для хранения данных, которые используются разными скриптами, однако определяются один раз. Например, к таким данным можно отнести константы, используемые в тест-кейсах, данные из тестируемого приложения, которые используются в различных частях приложения;

- для хранения данных, которые необходимо ввести в приложение перед началом тестирования. Например, прежде чем начать запуск тест-кейсов, необходимо добавить в тестируемое приложение записи, с которыми затем будут работать скрипты;

- для непосредственного доступа к базе данных тестируемого приложения. Например, для проверки того, что по определенному запросу в приложении выводится необходимый набор данных. Для этого можно считать данные из базы данных, а затем сравнить их с тем, что выводится при работе с приложением;

- для хранения результатов запущенных тест-кейсов. Например, вместо файлов результатов SilkTest, результаты тестирования можно выводить в базу данных для дальнейшего анализа (находить среднее время работы тест-кейсов, выделять шаги, на которые затрачено наибольшее время);

- при наличии среды, управляющей запуском тест-кейсов в определенное время без непосредственного участия человека, в базе можно хранить наборы тест-кейсов, предназначенные для различных видов тестирования, или для тестирования разных частей приложения.

Подобное хранение данных в одном месте упрощает их просмотр и редактирование.

Рассмотрим пример использования внешних данных из Excel-файла для тестирования приложения, представленного одной формой с компонентом ввода элемента поиска Search. Подключение данных из Excel-файла выполняет следующая последовательность шагов:

Шаг 1: создать Excel-файл с именем Search и заполнить Excel-таблицу данными для поиска (рисунок 6).

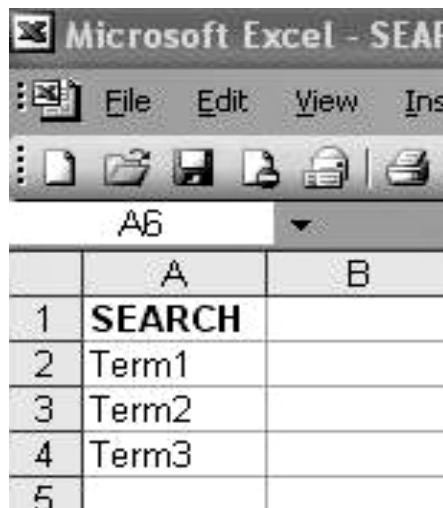
Шаг 2: зайти в SilkTest Window и выбрать пункт меню Tools.

Шаг 3: выбрать подпункт меню Data Driven Testcase (тест-кейс должен быть открыт в SilkTest).

Шаг 4: открыть окно Select Testcase. Выбрать тест-кейс и нажать «Ok» (рисунок 7).

Шаг 5: открыть окно Specify Data Driven Script. Выбрать «Create a new file/Overwrite an existing file» и нажать «Ok» (рисунок 8).

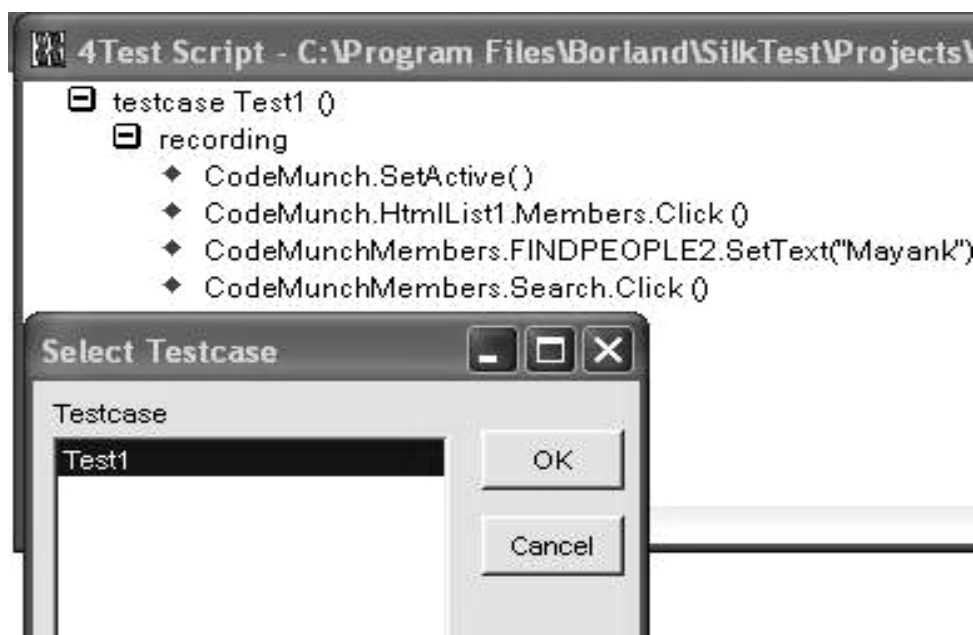




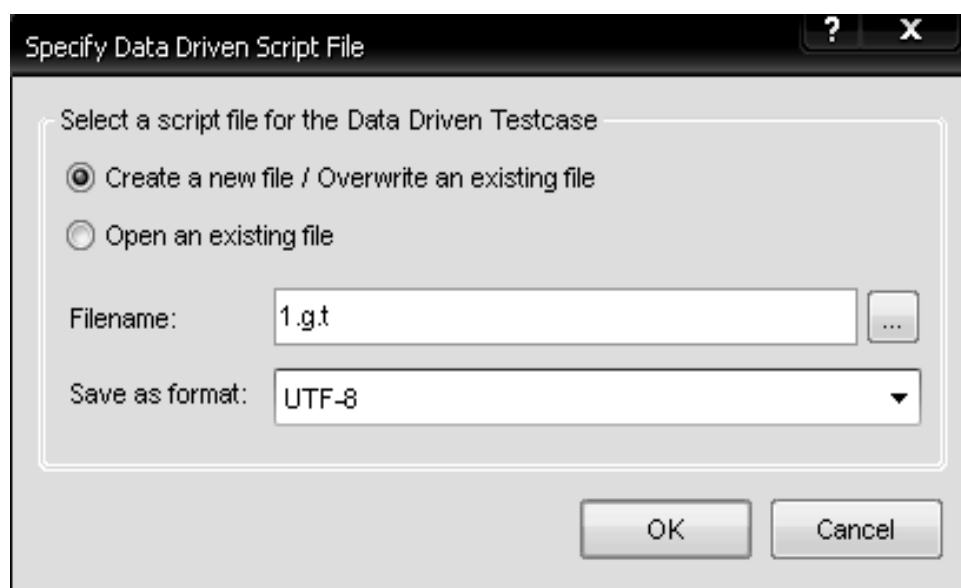
Microsoft Excel - SEARCH

	A	B
1	SEARCH	
2	Term1	
3	Term2	
4	Term3	
5		

**Рисунок 6** – Данные для поиска



**Рисунок 7** – Окно для выбора теста



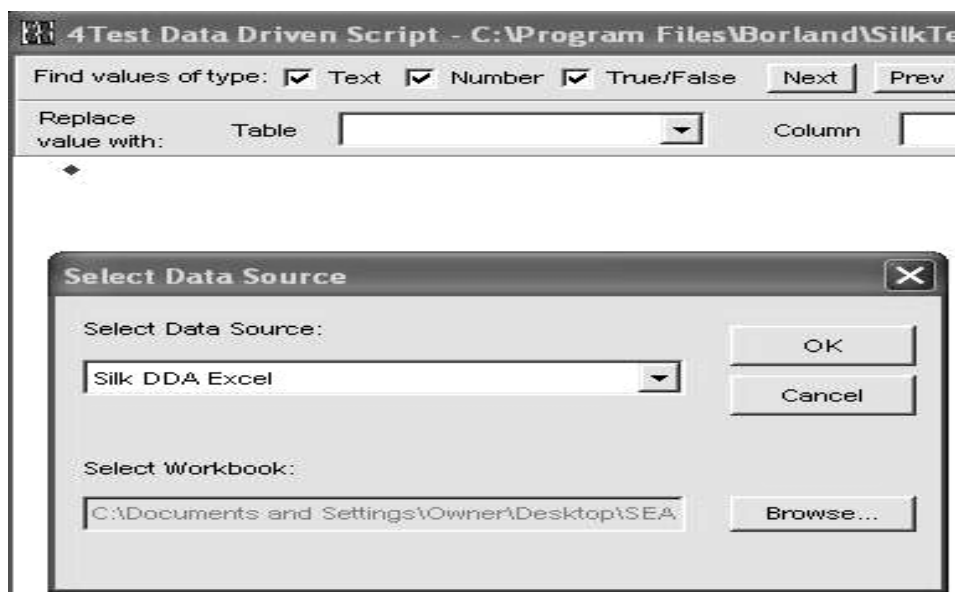
**Рисунок 8** – Создание нового файла

Шаг 6: открыть окно Select Data Source. Выбрать Silk DDA Excel. Выбрать Excel-файл через диалог Browse и нажать «Ok» (рисунок 9).

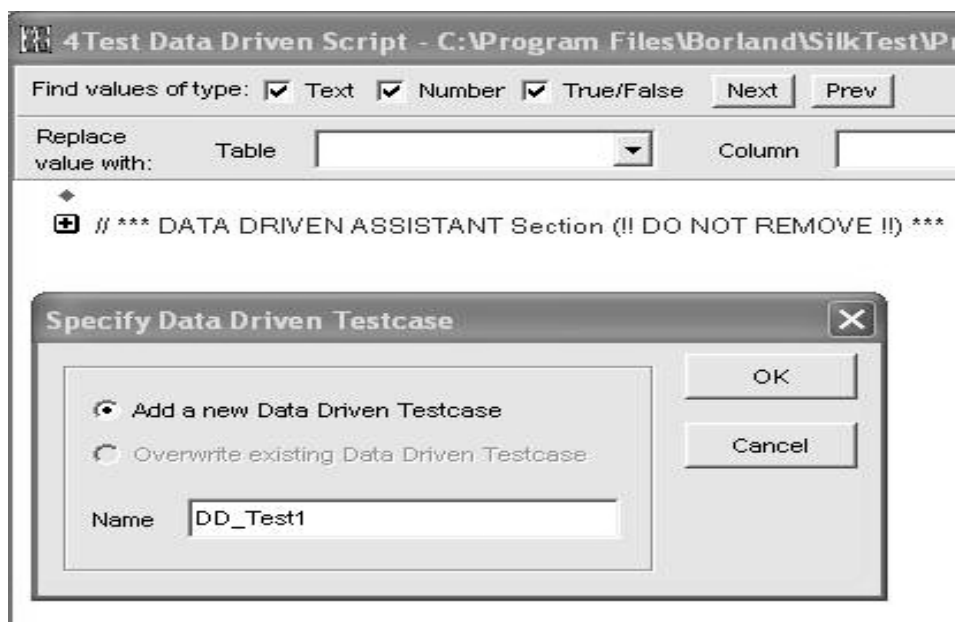
Шаг 7: открыть Specify Data Driven Testcase окно. Выбрать Add a new Data Driven Testcase и нажать «Ok» (рисунок 10).

Шаг 8: открыть окно Find/Replace Values. Нажать Cancel в Find/Replace Values (рисунок 11).

Шаг 9: войти в файл, сохраненный с именем скрипта, но с расширением \*.g.t (рисунок 12).



**Рисунок 9** – Выбор Excel-файла



**Рисунок 10** – Создание нового тест-кейса

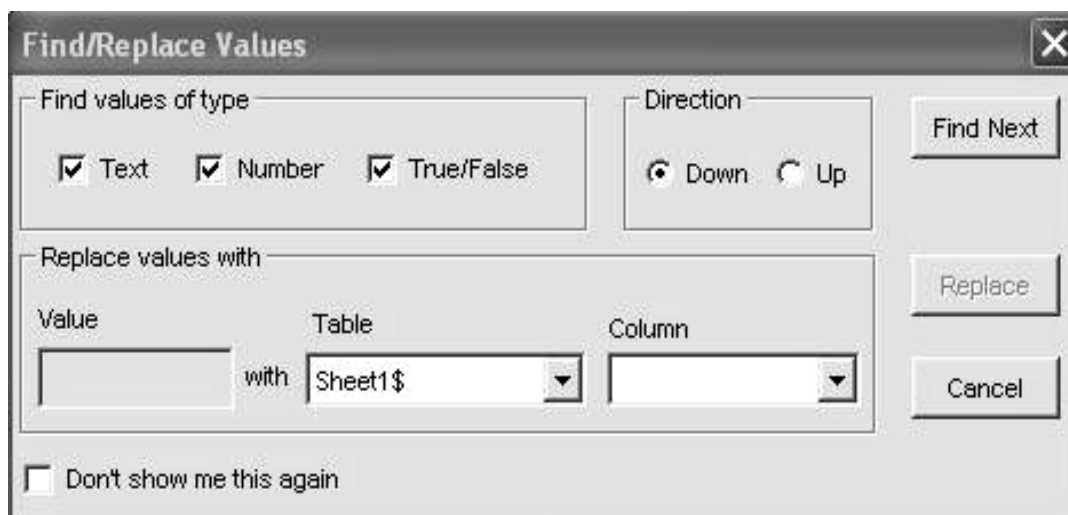
Шаг 10: в файле выбрать тип данных (Find values of type). В рассматриваемом примере тестовые данные представляют собой текст, поэтому следует выбрать Text. Возможные варианты выбора: текстовые

(Text), числовые (Number) или логические данные (True/False). В Replace value with для Table выбрать Sheet1\$, в Column выбрать столбец с именем, который присвоен столбцу в Excel-файле Search.

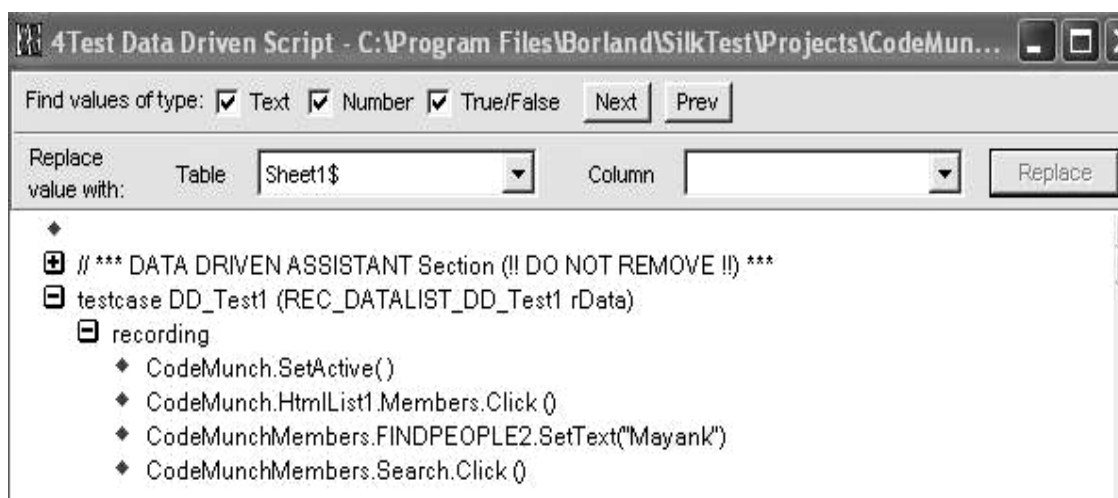
Шаг 11: используя кнопку Replace, вставить указанные значения в DD-скрипт (рисунок 13).

Шаг 12: нажать кнопку «Replace» (рисунок 14).

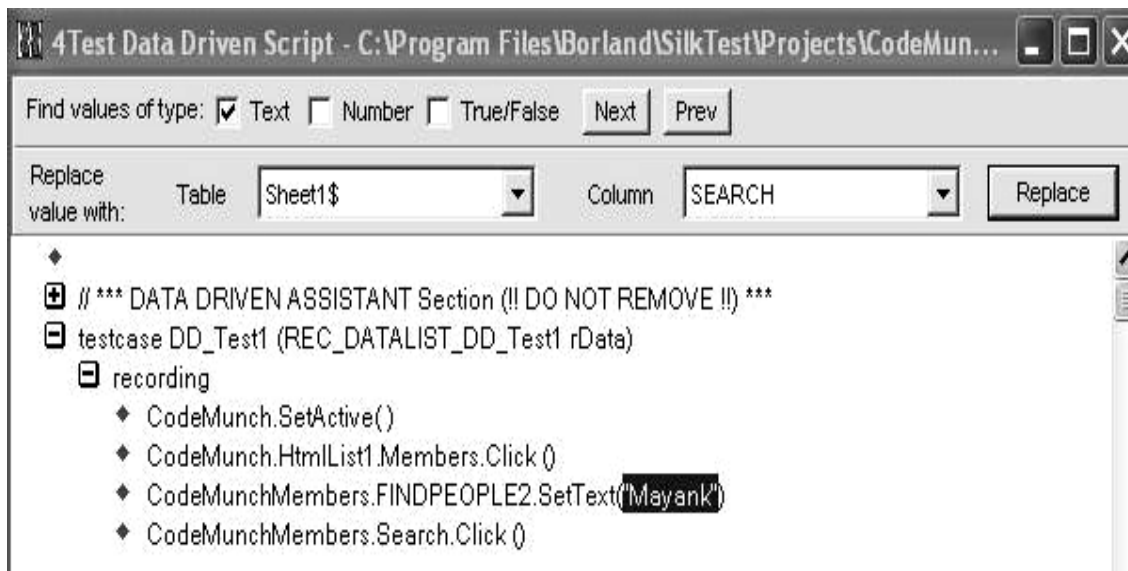
Шаг 13: нажать «Save» и выбрать пункт меню Run TestCase. В открывшемся окне выбрать скрипт и запустить его на выполнение.



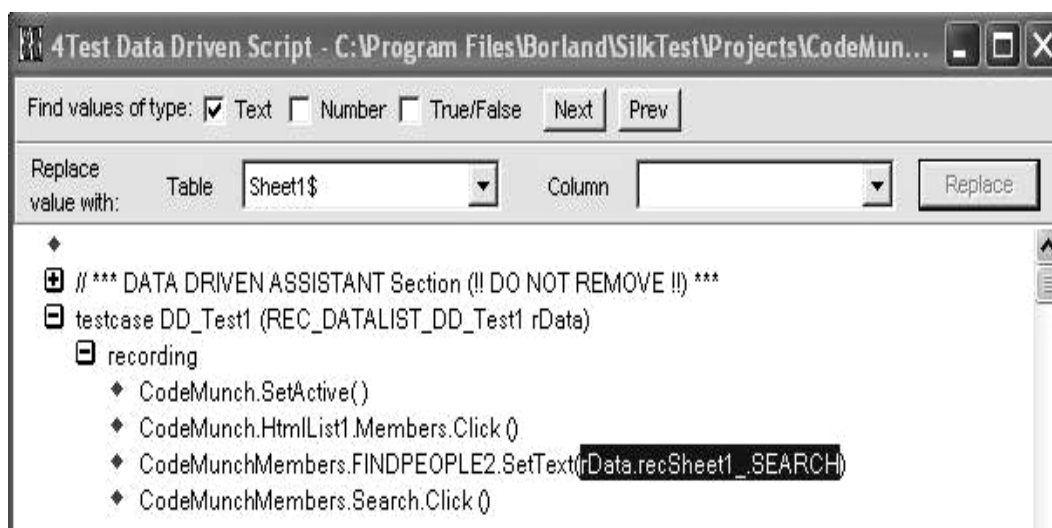
**Рисунок 11 – Поиск значений**



**Рисунок 12 – Открытие файла**



**Рисунок 13** – Добавление значения в DD-скрипт



**Рисунок 14** – Изменение скрипта

### ***1.5 Просмотр результатов исполнения тестовых сценариев***

После запуска тест-кейса, тестплана или набора тест-кейсов SilkTest выводит окно Results с результатами исполнения тест-кейсов. В нем отображается статистика по пройденным тест-кейсам (общее количество, количество и процент тест-кейсов, прошедших без ошибок – Passed, количество и процент прошедших с ошибками – Failed), время, затраченное на прохождение всех тест-кейсов.

В этом окне также отображается информация, выводимая в процессе работы скрипта функциями Print, ListPrint, LogError, LogWarning.

Если вы дочитали до этого момента, то вы умнички. Данная лабораторная – это проверка чтения методички. Далее читайте страницу двадцать восемь.

Для просмотра информации, выводимой в процессе работы тест-кейса, необходимо нажать на значок [+] слева от названия тест-кейса. В файле результатов (с расширением \*.res и таким же именем, как и название самого файла скрипта) хранятся результаты запусков.

В разделах Results и Debug можно изменить настройки вывода результатов, используя следующие опции:

- History size – количество хранящихся результатов, выполненных тесткейсов. По умолчанию хранятся только 5 последних результатов (значение по умолчанию может быть изменено до 32767);

- 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 завершения работы самого SilkTest. Однако это замедляет работу скрипта;

- 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;

- 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 тест-кейсам;

- Directory/File – здесь можно указать папку, в которую SilkTest будет 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 передаваемые им параметры), которые происходят во время работы скрипта;

- Print tags with agent calls – если включено, то при вызове методов будут выводиться и теги используемых окон.

- Пункты меню Results:

- Select – позволяет выбрать один из сохраненных результатов;

- 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 тест-кейса (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 тест-кейса, количество ошибок и их текст, количество предупреждений и т.п.) в текстовый файл с разделителями. Это может быть полезно для последующего преобразования 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 тест-кейса или скрипта (но не другого тестплана);

– Show Summary/Hide Summary – скрыть/показать суммарную информацию по каждому тест-кейсу;

– Goto Source – позволяет перейти к скрипту, откуда был вызван тест-кейс. Если курсор находится в файле результатов в конкретном тест-кейсе, то после открытия файла скрипта курсор будет помещен в начало этого тест-кейса. Если курсор в файле результатов находится на сообщении об ошибке, то при открытии файла скрипта курсор будет помещен в место, откуда было вызвано сообщение об ошибке;

– Mark Failures in Plan – выделяет в тестплане только те тест-кейсы, которые прошли с ошибками, после чего можно их перезапустить отдельно от остальных.

## ***1.6 Пример разработки теста***

### **Задание для тестирования**

Разработать тест-кейсы для проверки выполнения операции «Сложение» в двоичной системе счисления над 2-байтными операндами для стандартного тест-приложения «Калькулятор» вида «Программист».

#### **Тест-кейс №1**

Идея: операция сложения двух операндов в случае переполнения.

Шаги инструкции:

- 1 Запустить приложение.
- 2 Проверить запущено ли приложение.
- 3 Переключиться на вид «Инженерный».
- 4 Проверить переключение на вид «Инженерный».
- 5 Переключиться на вид «Программист».
- 6 Проверить переключение на вид «Программист».
- 7 Переключиться на операцию «Bin-2 байта».
- 8 Проверить переключение на операцию «Bin-2 байта».
- 9 Проверить доступность кнопок для двоичной системы счисления (доступны кнопки: 0, 1; недоступны кнопки: 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).
- 10 Ввести первый операнд (1111 1111 1111 1111).
- 11 Набрать знак операции «+».
- 12 Ввести второй операнд (1).
- 13 Нажать клавишу «=».
- 14 Сравнить результаты. **Ожидаемый результат: 0.**

#### **Тест-кейс №2**

Идея: операция сложения двух отрицательных операндов.

Шаги инструкции:

- 1 Запустить приложение.
- 2 Проверить запущено ли приложение.
- 3 Переключиться на вид «Инженерный».
- 4 Проверить переключение на вид «Инженерный».
- 5 Переключиться на вид «Программист».
- 6 Проверить переключение на вид «Программист».
- 7 Переключиться на операцию «Bin-2 байта».
- 8 Проверить переключение на операцию «Bin-2 байта».
- 9 Проверить доступность кнопок для двоичной системы счисления (доступны кнопки: 0, 1; недоступны кнопки: 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F);
- 10 Набрать первый операнд (1010).
- 11 Нажать клавишу +/- для смены знака.
- 12 Проверить результат (-1010).
- 13 Набрать знак операции «+».
- 14 Проверить перевод первого слагаемого в дополнительный код
- 15 (1111 1111 1111 0110).
- 16 Набрать второй операнд (1011).
- 17 Нажать клавишу +/- для смены знака.
- 18 Проверить результат (-1011).
- 19 Нажать клавишу «=».
- 20 Сравнить результаты. **Ожидаемый результат: 1111 1111 1110**

**1011.**

### **Тест-кейс №3 (Data Driven Testcase)**

Идея: операция сложения двух операндов.

Шаги инструкции:

- 1 Запустить приложение.
- 2 Проверить запущено ли приложение.
- 3 Переключиться на вид «Инженерный».
- 4 Проверить переключение на вид «Инженерный».
- 5 Переключиться на вид «Программист».
- 6 Проверить переключение на вид «Программист».
- 7 Переключиться на операцию «Bin-2 байта».
- 8 Проверить переключение на операцию «Bin-2 байта».
- 9 Проверить доступность кнопок для двоичной системы счисления (доступны кнопки: 0, 1; недоступны кнопки: 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).
- 10 Ввод первого операнда (X) (из Excel-файла).
- 11 Набрать знак операции «+».
- 12 Ввод второго операнда (Y) (из Excel-файла).
- 13 Нажать клавишу «=».
- 14 Проверить результат (Z) (из Excel-файла).

**Дополнительные требования к созданию тестов**

1 Разработать следующие тестовые планы (TestPlan):

- запуск Тест-кейса №1;
- запуск Тест-кейса №2;
- запуск Тест-кейса №3;
- запуск Тест-кейсов №1, 2, 3.

2 Тест-кейс №3 создать как Data Driven Testcase, используя Excel-файл для выбора тестовых данных.

Формат Excel-файла, приведен в таблице 1.

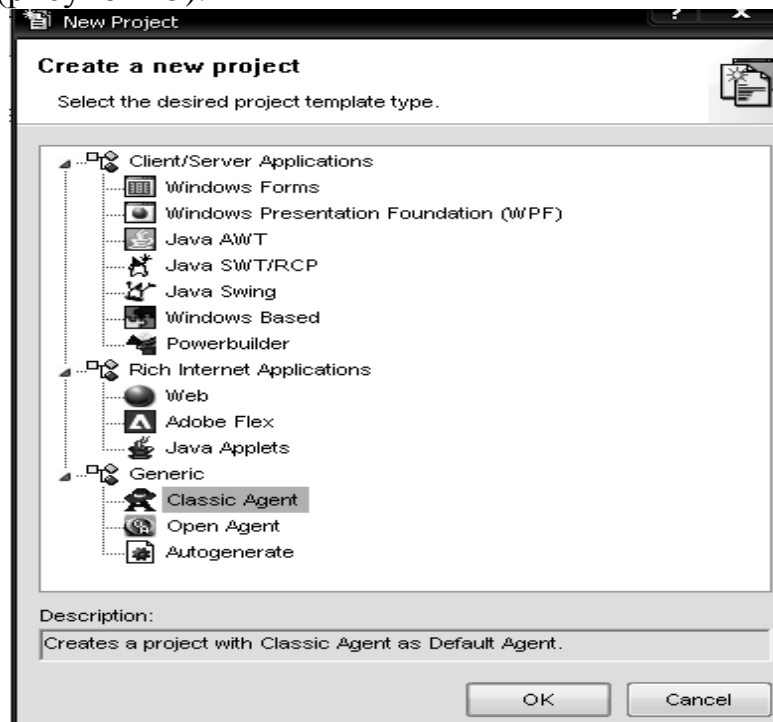
**Таблица 1** – Содержимое Excel-файла

Первый операнд X	Второй операнд Y	Ожидаемый результат Z
1000101	0111010	1111111

3 Шаги инструкций 1 и 2, общие для тест-кейсов №1, 2, 3, поместить в Recovery-файл.

### **Создание фрейма приложения**

На первом шаге необходимо создать новый проект, куда будут помещаться далее разработанные скрипты и тестпланы. Для этого в SilkTest следует выбрать пункт меню File-New Project и указать Classic Agent. Нажать «Ok» (рисунок 15).



**Рисунок 15** – Создание нового проекта

После задания имени проекта, например Add, откроется окно рисунок 16, в которое можно помещать описания окон и скрипты.

Теперь необходимо подготовить тест-приложение. Его можно открыть из главного меню Пуск-Программы-Стандартные-Калькулятор.

Далее следует приступить к записи описаний окон или оконных деклараций. Для этого в SilkTest необходимо выбрать пункт меню Record-



Window Declarations. После появления окна Record Window Declaration, навести курсор на заголовок окна «Калькулятор» и нажать комбинацию клавиш <Ctrl+Alt>. В результате окно Record Window Declaration примет вид, показанный на рисунок 17.



**Рисунок 16** – Задание имени проекта



**Рисунок 17** – Запись теста

Как видно среди описания окон выделено главное окно приложения. В разделе Window detail показано, что окно является окном класса MainWin с идентификатором «Калькулятор». В зависимости от того, к какому классу принадлежит окно или элемент управления, с ним можно выполнять разные действия. Далее представлено поле Tag. Тег (Tag) – это уникальная характеристика, по которому SilkTest распознает элементы управления в приложениях. В правой части окна Record Window Declaration можно просмотреть более подробную информацию о теге окна. В рассматриваемом примере тег состоит из заголовка (Caption) и идентификатора (Window ID).

Полученную декларацию окна уже можно вставлять в документ SilkTest. Однако при необходимости можно изменять настройки в системе распознавания окон (рисунок 18). Например, нажав кнопку «Options», отключить опцию Record Multiple Tags в нижней части окна Options, в разделе Default tag выбрать опцию Caption, в разделе Window declaration identifiers выбрать опцию Use the Caption.

Выполнив изменения настроек, следует нажать кнопку «Resume Tracking» (для возобновления записи) в окне Record Window Declaration. Далее необходимо навести курсор мыши на приложение «Калькулятор» и нажать клавиши <Ctrl+Alt>. В результате окно Record Window Declaration изменится. Теперь в качестве тега будет выступать только один идентификатор (например, Caption), а не несколько, как было в прошлый раз.

Нажав кнопку «Paste to Editor», можно вставить описание окна «Калькулятор» в документ SilkTest.

В полученном примере (рисунок 19) MainWin, Menu и MenuItem – это классы, по которым SilkTest различает, как работать с тем или иным элементом. Калькулятор, Правка, Вид, Справка – это имена объектов приложения. Имена объектов можно менять. Например, имя «Калькулятор» слишком длинно, его можно заменить на «Calc».

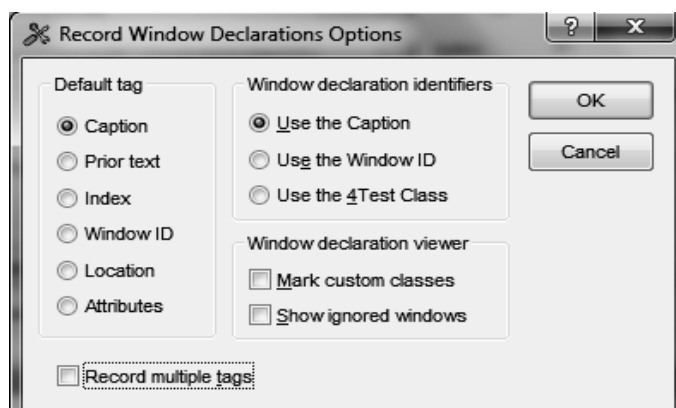


Рисунок 18 – Окно настроек

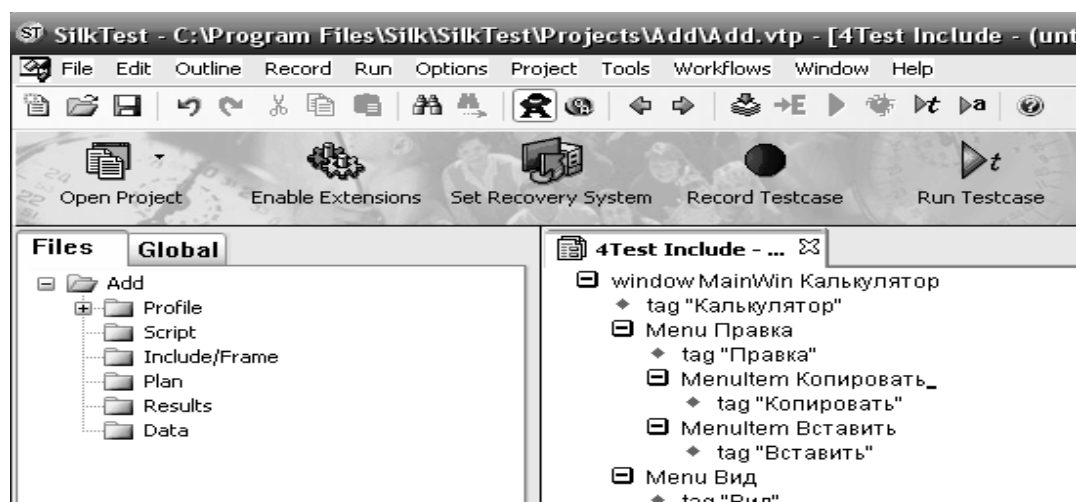


Рисунок 19 – Пример теста

## Настройка Recovery-системы

Для настройки Recovery-системы необходимо выбрать пункт «Set Recovery System». В появившемся окне выбрать тестируемое приложение «Калькулятор» и нажать кнопку «Ok» (рисунок 20). В результате чего в проект будет добавлен Recovery-файл, содержащий информацию о тестируемом приложении, в частности, вводимые константы, инициализируемые для дальнейшего использования в тест-кейсах (рисунок 21).

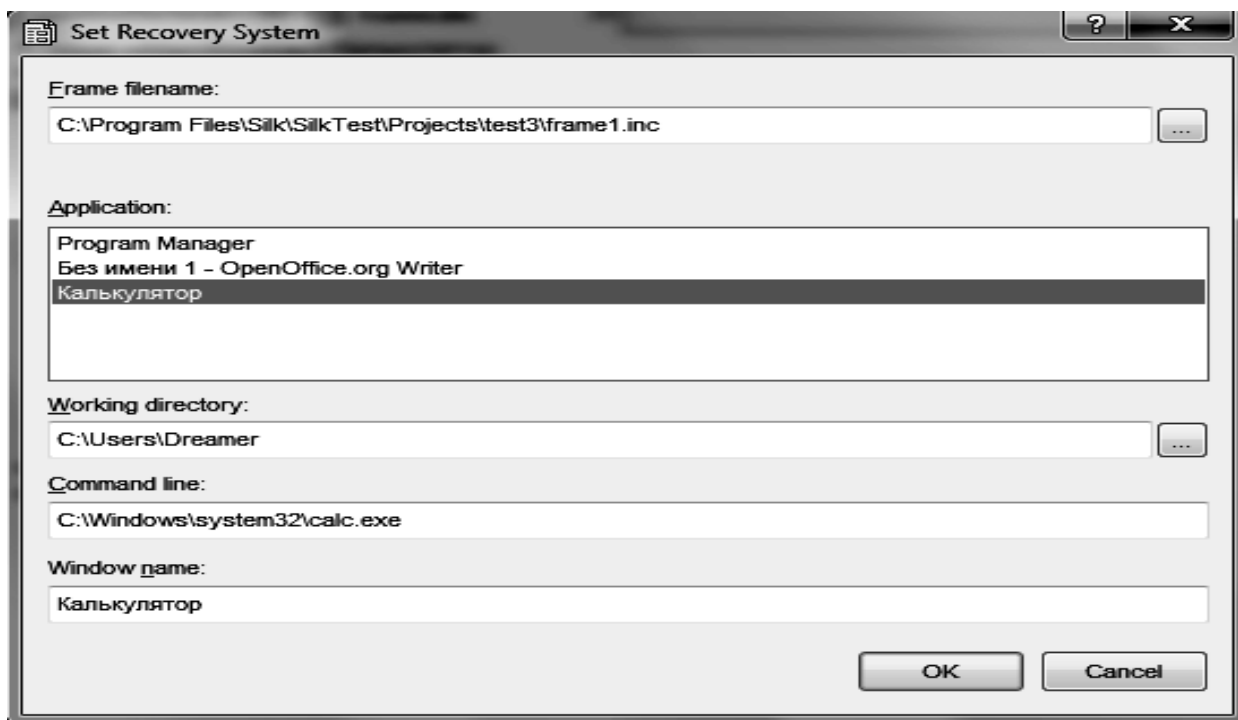


Рисунок 20 – Настройка Recovery-системы

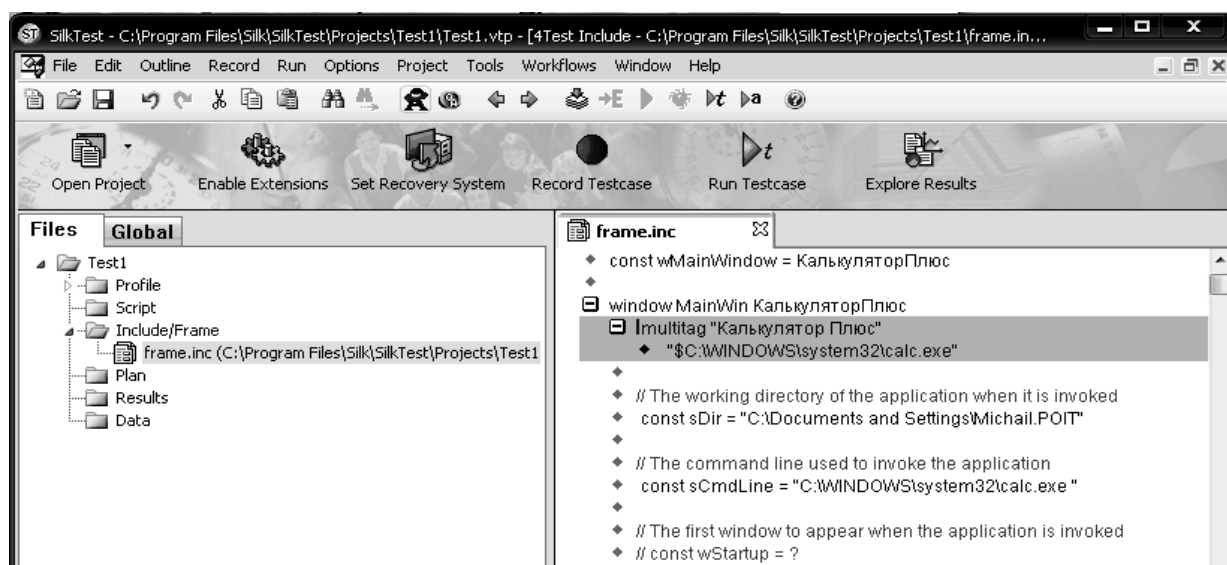


Рисунок 21 – Добавление Recovery-файла

После выполнения вышеописанных действий за запуск приложения в каждом тест-кейсе файла скрипта будет отвечать Recovery-система.

### **Запись и воспроизведение скрипта**

Когда оконные декларации созданы, следует приступить к записи и воспроизведению скрипта.

#### **Создание скрипта для тест-кейса №1**

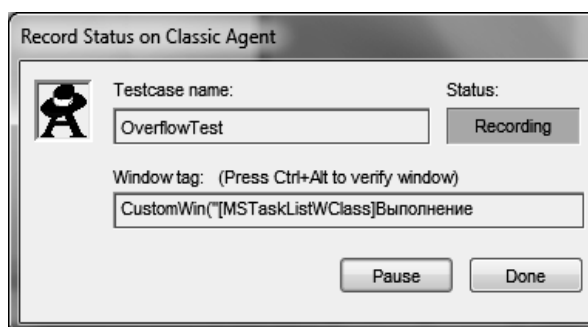
Для создания тест-кейса следует выбрать пункт меню Record-Testcase (рисунок 21). В поле Testcase name открывшегося окна ввести имя теста «Test1». Нажать кнопку «Start Recording» (рисунок 22).

На экране в правом нижнем углу появится окошко Record Status, отображающее статус записи и текущее окно под курсором мыши (рисунок 23).

Кнопка «Pause» позволяет приостановить запись теста, а кнопка «Done» остановить процесс записи теста.



**Рисунок 22 – Создание тест-кейса**

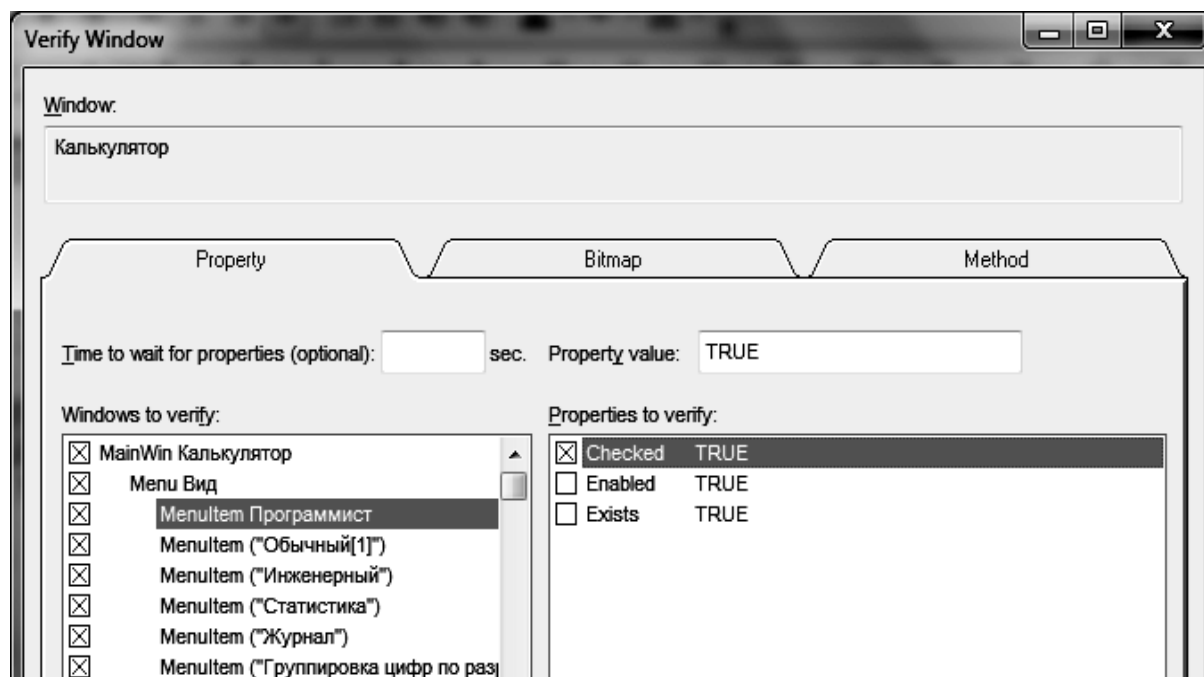


**Рисунок 23 – Задание имени теста**

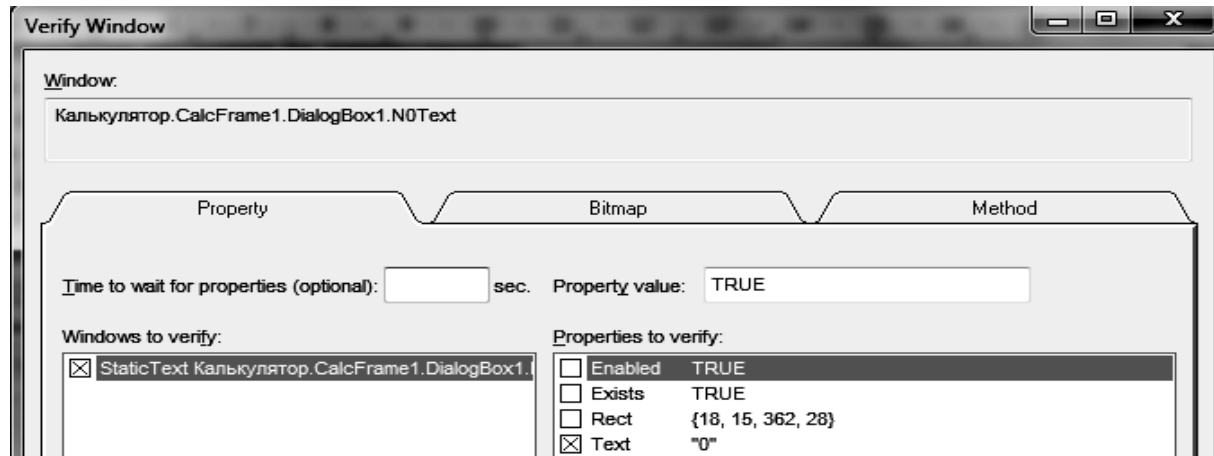
Находясь в режиме записи, выполним следующие действия:

- 1 Переключение приложения на вид «Программист».
- 2 Переключение на операцию «Bin-2 байта».
- 3 Проверка доступности кнопок ввода для двоичной системы счисления (доступны кнопки: 0, 1; недоступны кнопки: 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

Для проверки свойств объектов на главном окне приложения «Калькулятор» следует нажать сочетание клавиш <Ctrl+Alt>. После чего откроется окно Verify Properties приложения, где можно указать контролируемые свойства. Например, активность у приложения «Калькулятор» вида «Программист» (рисунок 24).



**Рисунок 24** – Указание свойств



**Рисунок 25** – Изменение свойств

Аналогично следует установить для проверки все требуемые состояния объектов приложения. После чего можно приступить к созданию кода для проверки операции сложения на переполнение:

- 1 Ввести первый операнд «1111 1111 1111 1111».
- 2 Выбрать операцию сложения «+».
- 3 Ввести второй операнд «1».
- 4 Нажать кнопку получения результата «=».
- 5 Проверить результат на равенство «0».

Для проверки результата над полем результата следует нажать сочетание <Ctrl+Alt> и установить проверку на «0» как показано на рисунке 26.

После выполнения всех действий необходимо завершить запись тест-кейса и выполнить генерацию кода путем нажатия кнопок «Done» и «Paste To Editor». В результате сгенерируется следующий программный код:

```
[ - ] testcase Test1 ()
[ - ] recording
//Активировать главное окно приложения Калькулятор
[ ] Калькулятор.SetActive ()
//Выбор в приложении Калькулятор пункта меню Вид->Программист
[ ] Калькулятор.Вид.Программист.Pick ()
//Выбор в приложении Калькулятор режима работы с бинарными операндами
[ ] Калькулятор.CalcFrame1.DialogBox2.RadioList1.Select ("Bin")
//Выбор режима работы с числами размера 2-Byte
[ ] Калькулятор.CalcFrame1.DialogBox2.BitMap3.Click (1, 16, 50)
//Активировать главное окно приложения Калькулятор
[ ] Калькулятор.SetActive ()
//Проверка начального состояния приложения Калькулятор
[ + ] Калькулятор.VerifyProperties ({...})
//Ввод первого операнда «1111 1111 1111 1111»
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
//Нажать кнопку «+»
[ ] Калькулятор.CalcFrame1.DialogBox2.N020.Click ()
//Ввод второго операнда «1»
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
//Нажать кнопку «=»
[ ] Калькулятор.CalcFrame1.DialogBox2.N022.Click ()
//Проверить числовое значение, содержащееся в окне результата
[ - ] Калькулятор.CalcFrame1.DialogBox1.N0Text.VerifyProperties ({...})
[ ] ""
[ - ] {...}
[ ] {"Text", "0"}
```

Проанализируем текст полученного программного кода. Скрипт начинается с ключевого слова `testcase`. Далее следует его имя (`Test1`), в скобках могут быть указаны передаваемые для этого тест-кейса параметры. Далее следует ключевое слово `recording`, показывающее, что код скрипта был записан автоматически. Это ключевое слово является информативным и его можно удалить. `Калькулятор.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()` для подтверждения. `Калькулятор.Программист.Pick()` – выбор пункта меню Вид-Инженерный. Метод `Pick()` используется для выбора пунктов меню (для других элементов управления обычно используется метод `Click()`, например, для нажатия цифр калькулятора). `Калькулятор.Move(x, y)` – метод `Move()` используется для передвижения окон по экрану. В качестве параметров в этот метод передаются новые координаты по горизонтали и вертикали. `Калькулятор.Close()` – метод `Close()` служит для закрытия окон.

Следует обратить внимание на то, как `SilkTest` обращается к объектам. Если нужно обратиться к пункту меню Инженерный, который является потомком меню Вид, нельзя написать `Калькулятор.Инженерный`. В этом случае `SilkTest` выдаст ошибку. В примере немного элементов управления и нет большой вложенности объектов (иерархии). Однако в больших приложениях, когда иерархия может достигать 10-20 уровней вложенности, возникает необходимость модификации фреймов с группировкой элементов управления, отличной от того, как это делает `SilkTest` по умолчанию.

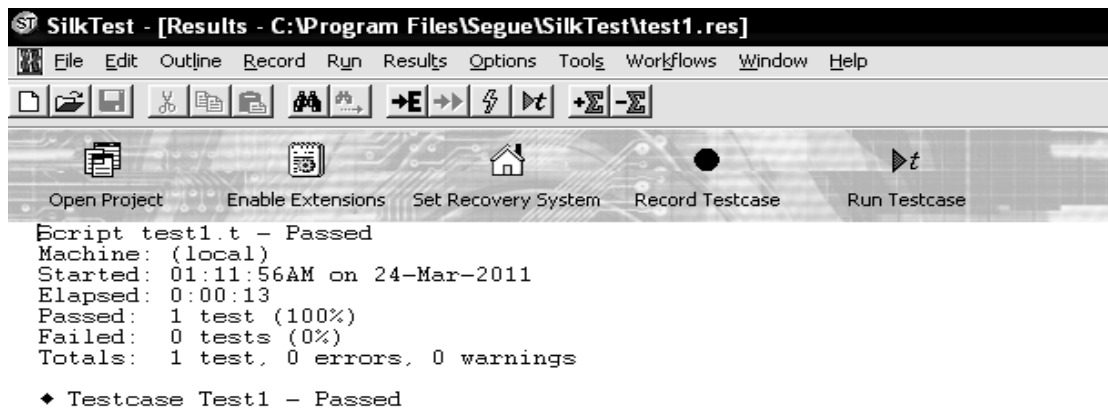
Заметим, что в данном коде следующая операция прописана несколько раз:

```
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click()
Модифицируем код, используя язык 4Test:
[-] testcase Test1 ()
    integer i
    [-] recording
    //Активировать главное окно приложения Калькулятор
        [ ] Калькулятор.SetActive ()
    //Выбор в приложении Калькулятор пункта меню Вид-Программист
        [ ] Калькулятор.Вид.Программист.Pick ()
    //Выбор в приложении Калькулятор режима работы с бинарными значениями
        [ ] Калькулятор.CalcFrame1.DialogBox2.RadioList1.Select ("Bin")
    //Выбор режима работы с числами размера 2-Byte
        [ ] Калькулятор.CalcFrame1.DialogBox2.BitMap3.Click (1, 16, 50)
    //Активировать главное окно приложения Калькулятор
        [ ] Калькулятор.SetActive ()
    //Проверить начальное состояние приложения Калькулятор
        [+] Калькулятор.VerifyProperties ({...})
    //Ввод первого операнда «1111 1111 1111 1111»
        for(i=0; i<16; i++)
            [ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
    //Нажать кнопку «+»
        [ ] Калькулятор.CalcFrame1.DialogBox2.N020.Click ()
```

```
//Ввод второго операнда «1»
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
//Нажать кнопку «=»
[ ] Калькулятор.CalcFrame1.DialogBox2.N022.Click ()
//Сравнить фактический результат с ожидаемым
[-] Калькулятор.CalcFrame1.DialogBox1.N0Text.VerifyProperties ({...})
[ ] ""
[-] {...}
[ ] {"Text", "0"}
```

Для того, чтобы запустить сгенерированный и отредактированный скрипт на выполнение необходимо выбрать пункт меню Run-Testcase. В появившемся окне указать тест-кейс «Test1» и нажать кнопку «Run».

Скрипт обрабатывает примерно за 1 секунду и SilkTest выдает отчет о проделанной работе: во сколько времени скрипт начал работать, как долго работал, сколько произошло ошибок и предупреждений (рисунок 26).



**Рисунок 26** – Отчет о проделанной работе

## Создание тест-кейса №2

Для создания тест-кейса №2, выполняющего проверку операции сложения над отрицательными числами, следует выполнить шаги по аналогии с тест-кейсом №1.

После выполнения шагов теста получим следующий код:

```
[-] testcase Test2 ()
[-] recording
//Активировать главное окно приложения Калькулятор
[ ] Калькулятор.SetActive ()
//Выбрать в приложении Калькулятор пункта меню Вид-Программист
[ ] Калькулятор.Вид.Программист.Pick ()
//Выбрать в приложении Калькулятор режим работы с бинарными значениями
[ ] Калькулятор.CalcFrame1.DialogBox2.RadioList1.Select ("Bin")
//Выбрать режим работы с числами размера 2-Byte
[ ] Калькулятор.CalcFrame1.DialogBox2.BitMap3.Click (1, 16, 50)
//Активировать главное окно приложения Калькулятор
[ ] Калькулятор.SetActive ()
//Проверить начальное состояние приложения Калькулятор
[+] Калькулятор.VerifyProperties ({...})
```



```

//Ввести первый операнд «1010»
    [ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click () // ввод 1
    [ ] Калькулятор.CalcFrame1.DialogBox2.N156.Click () // ввод 0
    [ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click () // ввод 1
    [ ] Калькулятор.CalcFrame1.DialogBox2.N156.Click () // ввод 0
//Нажать кнопку «+/-»
    [ ] Калькулятор.CalcFrame1.DialogBox2.N117.Click ()
//Проверить результат
    [-] Калькулятор.CalcFrame1.DialogBox1.N0Text.VerifyProperties ({...})
        [ ] ""
        [-] {...}
            [ ] {"Text",          "1111 1111 1111 0110"}
//Нажать кнопку «+»
    [ ] Калькулятор.CalcFrame1.DialogBox2.N020.Click ()
//Ввести второй операнд «1011»
    [ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click () // ввод 1
    [ ] Калькулятор.CalcFrame1.DialogBox2.N156.Click () // ввод 0
    [ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click () // ввод 1
    [ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click () // ввод 1
//Нажать кнопку «+/-»
    [ ] Калькулятор.CalcFrame1.DialogBox2.N117.Click ()
//Нажать кнопку «=»
    [ ] Калькулятор.CalcFrame1.DialogBox2.N022.Click ()
//Сравнить фактический результат с ожидаемым
    [-] Калькулятор.CalcFrame1.DialogBox1.N0Text.VerifyProperties ({...})
        [ ] ""
        [-] {...}
            [ ] {"Text",          "1111 1111 1111 0101"}

```

### Создание тест-кейса №3 как Data Driven Testcase

Для создания тест-кейса №3 следует использовать технологию Data Driven Test. Но сначала необходимо создать тест-кейс без использования Data Driven Test, выполнив шаги инструкции по аналогии с тест-кейсом №1. В результате будет сгенерирован следующий код:

```

[-] testcase Test3 ()
    [-] recording
//Инициализация переменных
    [ ] integer val1=1000101
    [ ] integer val2=111010
    [ ] integer result=1111111
    [ ] integer temp=10
    [ ] integer value
//Активировать главное окно приложения Калькулятор
    [ ] Калькулятор.SetActive ()
//Выбрать в приложении Калькулятор пункт меню Вид-Программист
    [ ] Калькулятор.Вид.Программист.Pick ()
//Выбрать в приложении Калькулятор режим работы с бинарными значениями
    [ ] Калькулятор.CalcFrame1.DialogBox2.RadioList1.Select ("Bin")
//Выбрать режим работы с числами размера 2-Byte

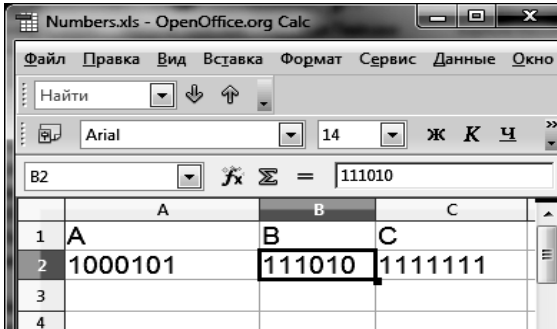
```

```

[ ] Калькулятор.CalcFrame1.DialogBox2.BitMap3.Click (1, 16, 50)
//Активировать главное окно приложения Калькулятор
[ ] Калькулятор.SetActive ()
//Проверить начальное состояние приложения Калькулятор
[+] Калькулятор.VerifyProperties ({...})
//Ввести в Калькулятор операнд val1, разбивая его на последовательность 1 и 0
[-] while(val1!=0)
    [ ] value= val1 % temp
    [-] if(value==1)
        [ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
    [-] if(value==0)
        [ ] Калькулятор.CalcFrame1.DialogBox2.N156.Click ()
    [ ] val1=(val1-value)/temp
//Нажать кнопку «+»
[ ] Калькулятор.CalcFrame1.DialogBox2.N020.Click ()
//Ввести в Калькулятор операнд val2, разбивая его на последовательность 1 и 0
[-] while(val2!=0)
    [ ] value= val2 % temp
    [-] if(value==1)
        [ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
    [-] if(value==0)
        [ ] Калькулятор.CalcFrame1.DialogBox2.N156.Click ()
    [ ] val2=(val2-value)/temp
//Нажать кнопку «=»
[ ] Калькулятор.CalcFrame1.DialogBox2.N022.Click ()
//Сравнить фактический результат с ожидаемым
[-] Калькулятор.CalcFrame1.DialogBox1.N0Text.VerifyProperties ({...})
[ ] ""
[-] {...}
    [ ] {"Text", Str(Result)}

```

Для создания Data Driven Test необходимо выполнить шаги, представленные на рисунке 27.



	A	B	C
1	A	B	C
2	1000101	111010	1111111
3			
4			

**Рисунок 27** – Таблица для тест-кейса

Шаг 1: создать файл с именем Numbers и заполнить Excel-таблицу данными для тест-кейса (рисунок 27).

Шаг 2: войти в SilkTest Window и выбрать пункт меню Tools.

Шаг 3: выбрать Data Driven Testcase.

Шаг 4: открыть окно Select Testcase. Выбрать тест-кейс Test3 (рисунок 7).

Шаг 5: открыть окно Specify Data Driven Script File. Выбрать «Create a new file/Overwrite an existing file» и нажать «Ok» (рисунок 8).

Шаг 6: открыть окно Select Data Source. Выбрать Silk DDA Excel. Выбрать Excel-файл через диалог Browse и нажать «Ok» после выбора (рисунок 9).

Шаг 7: открыть Specify Data Driven TestCase окно. Выбрать Add a new Data Driven Testcase и нажать «Ok».

Шаг 8: открыть окно Find/Replace Values. Нажать Cancel в Find/Replace Values (рисунок 11).

Шаг 9: войти в файл с именем скрипта, но с расширением \*.g.t (рисунок 12).

Шаг 10: в файле выбрать тип данных (Find values of type). Если тестовые данные представляют собой текст, выбрать Text. В Replace value with для Table выбрать Sheet1\$, в Column выбрать столбец с именем, который присвоен столбцу в Excel-файле. После выбора таблицы и столбца станет доступна кнопка Replace (рисунок 13).

Шаг 11: выбранные значения вставляются в скрипт (рисунок 14).

Последовательно выбирая данные для замены в скрипте и соответственно изменяя Column в таблице Excel, вставляются данные из DD-файла путем нажатия клавиши Replace для первого, второго операнда и ожидаемого результата. В результате описанных действий будет сгенерирован следующий программный код:

```
[ - ] testcase Test3 (REC_DATA_LIST_DD_Test3 rData)
//Инициализация переменных
[ ] integer val1=rData.recSheet1.A
[ ] integer val2=rData.recSheet1.B
[ ] integer result=rData.recSheet1.C
[ ] integer temp=10
[ ] integer value
//Активировать главное окно приложения Калькулятор
[ ] Калькулятор.SetActive ()
//Выбрать в приложении Калькулятор пункт меню Вид-Программист
[ ] Калькулятор.Вид.Программист.Pick ()
//Выбрать в приложении Калькулятор режим работы с бинарными значениями
[ ] Калькулятор.CalcFrame1.DialogBox2.RadioList1.Select ("Bin")
//Выбрать режим работы с числами размера 2-Byte
[ ] Калькулятор.CalcFrame1.DialogBox2.BitMap3.Click (1, 16, 50)
//Активировать главное окно приложения Калькулятор
[ ] Калькулятор.SetActive ()
//Ввести в Калькулятор операнд val1, разбивая его на последовательность 1 и 0
[ - ] while(val1!=0)
[ ] value= val1 % temp
[ - ] if(value==1)
[ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
[ - ] if(value==0)
```

```

        [ ] Калькулятор.CalcFrame1.DialogBox2.N156.Click ()
        [ ] val1=(val1-value)/temp
//Нажать кнопку «+»
        [ ] Калькулятор.CalcFrame1.DialogBox2.N022.Click ()
//Ввести в Калькулятор операнд val2, разбивая его на последовательность 1 и 0
        [-] while(val2!=0)
        [ ] value= val2 % temp
        [-] if(value==1)
            [ ] Калькулятор.CalcFrame1.DialogBox2.N155.Click ()
        [-] if(value==0)
            [ ] Калькулятор.CalcFrame1.DialogBox2.N156.Click ()
        [ ] val2=(val2-value)/temp
//Нажать кнопку «=»
        [ ] Калькулятор.CalcFrame1.DialogBox2.N022.Click ()
//Сравнить фактический результат с ожидаемым
        [-] Калькулятор.CalcFrame1.DialogBox1.N0Text.VerifyProperties ({...})
            [ ] ""
            [-] {...}
                [ ] {"Text", Str(result)}

```

### **Создание тест-плана**

Так как в проекте создано несколько тест-кейсов, их выполнение можно организовать в тестпланах по следующим сценариям:

- 1 Запуск тест-кейс №1.
- 2 Запуск тест-кейс №2.
- 3 Запуск тест-кейс №3.
- 4 Запуск тест-кейсов №1, 2, 3.

Для создания тестплана следует выбрать пункт меню File-New-Test Plan (рисунок 1). В результате будет создан пустой тестплан в виде файла с расширением \*.pln. Для добавления тест-кейсов в тестплан следует выбрать пункт TestPlan Detail.

Можете не выполнять задания из практической части, если Вы сдали предыдущие лабораторные до начала зачётной недели. Вместо отчёта сдаёте преподавателю пожелания на память. Условия действительно только на 14 неделе и на своей паре по расписанию.

Затем в поле Script выбрать файл-скрипт, содержащий необходимые тест-кейсы, а в поле Testcase выбрать из списка содержащихся в нем тест-кейсов необходимый. После чего нажать «Ok». В результате будет получен тестплан запуска тестов. Для запуска тестплана необходимо выбрать пункт Run-All Tests либо кнопку на панели управления SilkTest-Run the Script, suite or Testplan the Active Window.

## ОПИСАНИЕ ЯЗЫКА 4TEST

## П1.1. Типы данных

## П1.1.1. Синтаксис определения переменной

Синтаксис объявления переменной следующий:

**[scope] [share] data-type variable-id [expr],**

где *scope* – класс переменной. Может быть *public* (видимой для всех скриптов) или *private* (видимой только внутри определяющего скрипта); *share* – флаг определения доступности переменной несколькими процессами одновременно. Может быть *share*, если доступ разрешен, и *null* – если нет. Доступ контролируется оператором *access*; *data-type* – тип данных (таблица П1); *variable-id* – имя переменной; *expr* – инициализирующее выражение.

**Таблица П1 – Типы данных**

Тип данных	Описание	Граничные значения		Комментарии
		Минимальное значение	Максимальное значение	
INTEGER	Целое число	-2 147 483 648	2 147 483 647	
REAL	Дробь	(+/-) 2.23E-308	(+/-) 1.79E308	Машинозависим
NUMBER	Число	-	-	Объединение INTEGER и REAL
DATE	Дата	-4713.01.01 (отрицательные значения для дат до нашей эры)	19999.12.31 (положительные значения для дат нашей эры)	
TIME	Время	00:00:00.000000	23:59:59.999999	
STRING	Строка	Пустая строка	16 383 символов	
STRING constant	Строка-константа	Пустая строка	4 096 символов	
BOOLEAN	Логическая	-	-	TRUE или FALSE переменная
Identifiers	Указатель	1 символ	2048 символов	

## П1.1.2. Синтаксис определения константы

Синтаксис объявления константы:

**[scope] const [data-type] const-name=expr,**

где *scope* – класс переменной. Может быть *public* (видимой для всех скриптов) или *private* (видимой только внутри определяющего скрипта); *data-type* – тип данных (INTEGER, REAL, STRING или BOOLEAN); *const-name* – имя константы; *expr* – инициализирующее выражение.

## П1.1.3. Примеры определения и инициализации переменных

Пример 1. Инициализация переменной *iVarNum* как целочисленной переменной со значением 0 и доступной для нескольких процессов одновременно:

```
public share integer iVarNum=0
```

Пример 2. Инициализация переменной User\_report как строки со значением «Step №7» и видимой для всех скриптов и недоступной более чем одному потоку:

```
public string User_report="Step №7"
```

Пример 3. Инициализация переменной User\_report как строчной константы со значением «Step №7»:

```
public const string User_report="Step №7"
```

#### П1.1.4. Массивы данных

4Test поддерживает только жестко определенные массивы с возможностью изменения количества элементов. Массивы могут быть как одномерными, так и многомерными. Объявление массива имеет следующий синтаксис:

**ARRAY [dimension] [,dimension] OF data-type array-id ,**

где dimension – имя массива; data-type – тип данных массива; array-id – имя массива.

Обращение к элементам такого массива осуществляется посредством указания имени массива и порядкового номера элемента (первый элемент имеет индекс 1).

Частным случаем массива является список (LIST). Список может состоять из данных любого типа и отличается от массива тем, что границы списка не определяются при его объявлении. Объявления массива-списка имеют следующий синтаксис:

**LIST [OF data-type] list-id [=elements] ,**

где data-type – тип данных массива; list-id – имя списка; elements – элементы списка.

Обращение к элементам такого списка осуществляется посредством указания имени списка и порядкового номера элемента (первый элемент имеет индекс 1).

### П1.2. Операторы и функции языка 4Test

#### П1.2.1. Арифметические операторы

Арифметические операторы, поддерживаемые 4Test, представлены в таблице П2.

**Таблица П2 – Арифметические операторы**

Оператор	Значение	Оператор	Значение
+	Сложение	%	Деление нацело (остаток от деления)
—	Вычитание	**	Возведение в степень
*	Умножение	++	Увеличение на 1
/	Деление	--	Уменьшение на 1

### П1.2.2. Логические операторы

Логические операторы, поддерживаемые 4Test, представлены в таблице П3.

**Таблица П3 – Логические операторы**

Оператор	Значение
&&	И
	ИЛИ
!	Не

### П1.2.3. Операторы сравнения

Операции отношения, поддерживаемые 4Test, представлены в таблице П4.

**Таблица П4 – Операции отношения**

Оператор	Значение
>	Больше
>=	Больше или равно
<	Меньше
<=	Меньше или равно
==	Равно
!=	Не равно

### П1.2.4. Условные операторы

4Test поддерживает условные операторы IF ELSE и SWITCH. Условный оператор IF... ELSE имеет следующий синтаксис:

```
if boolean-expr  
  statement
```

```
[else  
  statement]
```

Условный оператор SWITCH имеет следующий синтаксис:

```
switch (expr)  
  case case-value(s)  
    statement  
  [case case-value(s)  
    statement]...  
  [default  
    statement]
```

### П1.2.5. Операторы цикла

4Test поддерживает операторы цикла FOR и WHILE. Оператор цикла FOR имеет следующий синтаксис:

```
for ([init-stmt] : [boolean-expr] ; [incr-stmt]) statements
```

4Test поддерживает два варианта оператора цикла FOR.

Первый «for numeric iterations»:

```
for loop-var=start-expr to end-expr [step step-expr] statement
```

Второй «for each»:

**for each item in expr statement**

Оператор цикла WHILE имеет следующий синтаксис:

**while boolean-expr statement**

#### П1.2.6. Функции приведения типа

Функции приведения типа языка 4Test представлены в таблице П5.

**Таблица П5 – Функции приведения типа**

Наименование функции	Выполняемое действие	Параметры функции
<i>sNum=Str(nNum [, iWidth, iDec])</i>	Функция приводит переменную типа NUMBER, содержащую цифровое представление символьного значения, в переменную типа STRING, содержащую соответствующее символьное значение	<i>sNum</i> – переменная типа STRING, полученная путем приведения <i>nNum</i> ; <i>nNum</i> – переменная типа NUMBER, содержащая цифровое представление символьного значения; <i>iWidth</i> – длина возвращаемой строки (необязательный параметр); <i>iDec</i> – количество цифр после запятой (необязательный параметр)
<i>nNum=Val(sToConvert)</i>	Функция приводит переменную типа STRING, содержащую символьное представление цифрового значения, в переменную типа NUMBER, содержащую соответствующее цифровое значение	<i>nNum</i> – переменная типа NUMBER, полученная приведением <i>sToConvert</i> ; <i>sToConvert</i> – переменная типа STRING, содержащая символьное представление цифрового значения
<i>iCode =Asc (sString)</i>	Функция возвращает ASCII-код символа <i>sString</i>	<i>sString</i> - символ
<i>sChar = Chr (iInt)</i>	Функция возвращает символ со значением ASCII-кода, равным <i>iInt</i>	<i>iInt</i> - ASCII-код

#### П1.2.7. Функции работы с массивами

Функции работы с массивами языка 4Test представлены в таблице П6.

**Таблица П6 – Функции работы с массивами**

Наименование функции	Выполняемое действие	Параметры функции
<i>ArrayFind (aArray, aElem [, iMaxIndex])</i>	Функция возвращает индекс элемента <i>aElem</i> в массиве <i>aArray</i> (0, если элемент не найден)	<i>aArray</i> – имя массива; <i>aElem</i> – искомый элемент; <i>iMaxIndex</i> – максимальное количество элементов, включенных в поиск
<i>ArrayResize(aArray, iNewSize [, iDim])</i>	Функция изменяет заданные размеры массива	<i>aArray</i> – имя массива; <i>iNewSize</i> – новое количество элементов; <i>iDim</i> – измерение массива
<i>ArraySize(aArray [, iDim])</i>	Функция возвращает количество элементов массива	<i>aArray</i> – имя массива; <i>iDim</i> – измерение массива
<i>ArraySort(aArray [, iMaxIndex])</i>	Функция сортирует массив	<i>aArray</i> – имя массива; <i>iMaxIndex</i> – максимальное количество элементов, включенных в сортировку



Функции работы со списками языка 4Test представлены в таблице П7.

**Таблица П7 – Функции работы со списками**

Наименование функции	Выполняемое действие	Параметры функции
<i>ListAppend (IList, aElem)</i>	Функция добавляет значение <i>aElem</i> в список <i>IList</i>	<i>IList</i> – имя списка; <i>aElem</i> – искомый элемент
<i>ListCount (IList)</i>	Функция возвращает количество элементов в списке.	<i>IList</i> – имя списка
<i>ListDelete (IList, ilndex)</i>	Функция удаляет указанный элемент списка	<i>IList</i> – имя списка; <i>ilndex</i> – искомый элемент
<i>ListInsert(IList, index, aElem)</i>	Функция вставляет значение <i>aElem</i> в позицию <i>index</i> списка <i>IList</i>	<i>IList</i> – имя списка; <i>index</i> – позиция изменения элемента; <i>aElem</i> – вставляемый элемент
<i>ListMerge(lOrig, lMerge [, iMergePos])</i>	Функция объединяет списки <i>lOrig</i> и <i>lMerge</i>	<i>lOrig</i> – имя первого списка; <i>lMerge</i> – имя второго списка; <i>iMergePos</i> – номер элемента первого списка, в который вставляется второй список
<i>ListFind (IList, altem)</i>	Функция возвращает индекс элемента <i>altem</i> в списке <i>IList</i> (0, если элемент не найден)	<i>IList</i> – имя списка; <i>altem</i> – искомый элемент
<i>ListRead(IList, sFileName)</i>	Функция считывает значения из файла <i>sFileName</i> в список <i>IList</i>	<i>IList</i> – имя списка; <i>sFileName</i> – имя файла
<i>ListSort (IList)</i>	Функция сортирует список <i>IList</i>	<i>IList</i> – имя списка
<i>ListPrint (IList)</i>	Функция распечатывает список <i>IList</i> в файл результатов	<i>IList</i> – имя списка

#### П1.2.6. Функции работы со строками

Функции работы со строками языка 4Test представлены в таблице П8.

**Таблица П8 – Функции работы со строками**

Наименование функции	Выполняемое действие	Параметры функции
<i>IsAlpha (sString)</i>	Функция проверяет, является ли первый символ в строке буквенным значением	<i>sString</i> – имя строки
<i>IsDigit (sString)</i>	Функция проверяет, является ли первый символ в строке цифровым значением	<i>sString</i> – имя строки
<i>IsSpace (sString)</i>	Функция проверяет, является ли первый символ в строке символом пробел	<i>sString</i> – имя строки
<i>GetField(sString, sDelim, iField)</i>	Функция возвращает сегмент строки по заданным параметрам.	<i>sString</i> – имя строки; <i>sDelim</i> – флаг разбиения (символ, на основе которого будет произведена разбивка); <i>iField</i> – номер возвращаемого сегмента
<i>Left(sString, iNumChars)</i>	Функция возвращает заданное количество символов, начиная от	<i>sString</i> – имя строки; <i>iNumChars</i> – количество возвращаемых

Наименование функции	Выполняемое действие	Параметры функции
	самого левого символьного значения	символов
<i>Right(sString, iNumChars)</i>	Функция возвращает заданное количество символов, начиная от самого правого символьного значения	<i>sString</i> – имя строки; <i>iNumChars</i> – количество возвращаемых символов
<i>Ltrim (sToStrip)</i>	Функция возвращает строку с удаленными начальными пробелами	<i>sToStrip</i> – возвращаемая строка
<i>Rtrim (sToStrip)</i>	Функция возвращает строку с удаленными конечными пробелами	<i>sToStrip</i> – возвращаемая строка
<i>Trim (sToStrip)</i>	Функция возвращает строку с удаленными пробелами	<i>sToStrip</i> – возвращаемая строка
<i>Space (iCount)</i>	Функция возвращает строку с заданным количеством пробелов	<i>iCount</i> – число
<i>Tabs(iAmount)</i>	Функция возвращает строку с заданным количеством табуляций	<i>iAmount</i> – число
<i>Replicate (sOrig, iCopies)</i>	Функция возвращает значение строки заданное количество раз	<i>sOrig</i> – искомая строка; <i>iCopies</i> – число
<i>Lower (string)</i>	Функция возвращает данную строку в нижнем регистре	<i>string</i> – имя строки
<i>Upper (sToConvert)</i>	Функция возвращает данную строку в верхнем регистре	<i>string</i> – имя строки
<i>MatchStr (sPattern, sString)</i>	Функция проверяет строку на присутствие «маски» поиска	<i>sPattern</i> – «маска» поиска; <i>sString</i> – искомая строка
<i>SubStr (sString, iPos [, iLen])</i>	Функция возвращает часть данной строки длиной <i>iLen</i> , считая от <i>iPos</i>	<i>sString</i> – данная строка; <i>iPos</i> – позиция в данной строке; <i>iLen</i> – длина вырезаемой части
<i>StrPos (sSubstr, sTarget [, bBackward])</i>	Функция возвращает положение <i>sTarget</i> в <i>sSubstr</i> (0, если строка не найдена)	<i>sSubstr</i> – строка, в которой производится поиск; <i>sTarget</i> – строка, которую ищут; <i>bBackward</i> – флаг начала поиска (TRUE, если поиск производится с конца, не указывается, если с начала)
<i>StrTran (sOrig, sSearch, sReplace)</i>	Функция заменяет <i>sSearch</i> на <i>sReplace</i> в <i>sOrig</i>	<i>sOrig</i> – данная строка; <i>sSearch</i> – искомая строка; <i>sReplace</i> – строка-замена
<i>Stuff (sOrig, iPos, iLen, sReplace)</i>	Функция вставляет <i>sReplace</i> в <i>sOrig</i> , начиная с <i>iPos</i> , удалив при этом <i>iLen</i> символов, начиная с <i>iPos</i>	<i>sOrig</i> – данная строка; <i>iPos</i> – начальная позиция изменения; <i>iLen</i> – количество удаляемых символов; <i>sReplace</i> – вставляемые значения
<i>Len(sString)</i>	Функция возвращает длину строки	<i>sString</i> – имя строки

## ЭЛЕМЕНТЫ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ И ИХ ФУНКЦИИ

## П2.1. Основные элементы графического интерфейса

Основные элементы графического интерфейса:

- Window (Окно);
- Menu (меню);
- ListBox/ComboBox (Раскрывающийся список/Поле со списком);
- Button (Кнопка);
- CheckBox/RadioButton (Флажок/Переключатель);
- EditBox (Поле ввода);
- Table (Таблица);
- Tab (Вкладка).

## П2.2. Функции элементов графического интерфейса

## П2.2.1. Функции для Window (окно)

Функции элемента Window:

- активирование окна;
- сворачивание окна;
- разворачивание окна;
- восстановление первоначального вида окна.

## П2.2.1.1. Активирование окна

Когда одновременно работают два и более приложений, бывает необходимо активировать окно тестируемого приложения, то есть расположить его поверх остальных окон:

***Windowname.SetActive()***, где *Windowname* - логическое имя окна.

Пример. Следующая строка кода активирует окно Window\_1:

*Window\_1.SetActive()*

## П2.2.1.2. Сворачивание окна

Когда одновременно открыто несколько окон тестируемого приложения, бывает необходимо свернуть одно окно и активировать другое:

***Windowname.Minimize()***, где *Windowname* – логическое имя окна.

Пример. Сворачивание окна Window\_1:

*Window\_1.Minimize()*

## П2.2.1.3. Разворачивание окна

Иногда бывает необходимо развернуть ранее свернутое окно:

***Windowname.Maximize()***, где *Windowname* – логическое имя окна.

Пример. Разворачивание окна Window\_1:

*Window\_1.Maximize()*

### П2.2.2. Функции для Menu (меню)

Элемент Menu - раскрывающийся список параметров, позволяющих выполнить то или иное действие. Функция этого элемента – выбор любого элемента из списка. Выбор осуществляется функцией:

***Menu.MenuItem.Pick()*** , где *Menu* — логическое имя меню; *MenuItem* – логическое имя параметра.

Пример. Следующая строка кода выбирает элемент *Option1*:

*Menu1.Option1.Pick()*

### П2.2.3. Функции для ListBox/ComboBox (Раскрывающийся список/ Поле со списком)

Для работы с элементом ListBox/ComboBox предусмотрены следующие возможности:

- выбор элемента списка по его значению;
- выбор элемента списка по его порядковому номеру;
- выбор нескольких элементов списка по их значениям;
- выбор нескольких элементов списка по их порядковым номерам;
- определение порядкового номера элемента списка по его значению;
- определение значения элемента списка по его порядковому номеру;
- определение количества элементов в списке;
- определение значений всех элементов в списке;
- определение значения выбранного элемента списка.

#### П2.2.3.1. Выбор элемента списка по его значению

Выбор осуществляется функцией:

***ComboBox/ListBox.Select(item)*** , где *ComboBox/ListBox* – логическое имя списка; *item* – выбираемый элемент.

Пример. Выбор элемента Visa из списка CreditCard = (Visa, Master Card, American Express):

*CreditCard.Select("Visa")*

#### П2.2.3.2. Выбор элемента списка по его порядковому номеру

Выбор осуществляется функцией:

***ComboBox/ListBox.Select(item)*** , где *ComboBox/ListBox* – логическое имя списка; *item* – выбираемый элемент.

Пример. Строка кода для выбора элемента Visa из списка Credit Card = (Visa, Master Card, American Express):

*CreditCard. Select ("#0")*

#### П2.2.3.3. Выбор нескольких элементов списка по их значениям

Выбор осуществляется функцией:

***ComboBox/ListBox.MultiSelect(sItem)*** , где *ComboBox/ListBox* — логическое имя списка; *sItem* - выбираемый элемент.

Пример. Следующий блок кода выбирает элемент January из списка Month:

*Month.MultiSelect("January")*

Для обеспечения множественного выбора, необходимо использовать функцию несколько раз.

П2.2.3.4. Выбор нескольких элементов списка по их порядковым номерам

Выбор осуществляется функцией:

***ComboBox/ListBox.MultiSelect(sItem)*** , где *ComboBox/ListBox* – логическое имя списка; *sItem* – выбираемый элемент.

Пример. Следующий блок кода выбирает элемент January из списка Month:

*Month.MultiSelect("#0")*

Для обеспечения множественного выбора, необходимо использовать функцию несколько раз.

П2.2.3.5. Определение порядкового номера элемента списка по его значению

Определение осуществляется функцией:

***ComboBox/ListBox.FindItem(sItem)*** , где *ComboBox/ListBox* – логическое имя списка; *sItem* – значение элемента.

Пример. Строка кода, возвращающая порядковый номер элемента Visa из списка CreditCard= (Visa, Master Card, American Express):

*out\_value = CreditCard.FindItem("Visa")*

П2.2.3.6. Определение значения элемента списка по его порядковому номеру

Определение осуществляется функцией:

***ComboBox/ListBox.GetItemText(sItem)*** , где *ComboBox/ListBox* – логическое имя списка; *sItem* – порядковый номер элемента.

Пример. Строка кода, возвращающая значение элемента с порядковым номером 1 из списка CreditCard = (Visa, Master Card, American Express):

*out\_value = CreditCard.GetItemText("#0")*

П2.2.3.7. Определение количества элементов в списке

Определение осуществляется функцией:

***ComboBox/ListBox.GetItemCount()*** , где *ComboBox/ListBox* – логическое имя списка.

Пример. Подсчет количества элементов в списке CreditCard = (Visa, Master Card, American Express):

*out\_value = CreditCard.GetItemCount()*

П2.2.3.8. Определение значений всех элементов в списке

Определение осуществляется функцией:

***ComboBox/ListBox.GetContents()*** , где *ComboBox/ListBox* – логическое имя списка.

Пример. Следующая строка кода возвращает значения всех элементов в списке CreditCard = (Visa, Master Card, American Express):

```
out_value = CreditCard.GetContents()
```

#### П2.2.3.9. Определение значения выбранного элемента списка

Определение осуществляется функцией:

***ComboBox/ListBox.ReturnType()*** , где *ComboBox/ListBox* - логическое имя списка; *ReturnType* – функция возврата данных; *GetSelIndex* – порядковый номер выбранного элемента; *GetSelText* – значение выбранного элемента.

Пример 1. Строка кода, возвращающая порядковый номер выбранного элемента из списка CreditCard = (Visa, Master Card, American Express):

```
out_num = CreditCard.GetSelIndex()
```

Пример 2. Строка кода, возвращающая значение выбранного элемента из списка CreditCard = (Visa, Master Card, American Express):

```
out_value = CreditCard.GetSelText()
```

#### П2.2.4. Функции для Button (кнопка)

Для элемента Button предусмотрены следующие функции:

- нажатие кнопки;
- получение сведений о состоянии элемента.

##### П2.2.4.1. Нажатие кнопки

Нажатие кнопки реализует следующая функция:

***PushButton.Click()*** , где *PushButton* – логическое имя кнопки.

Пример. Нажатие кнопки Submit:

```
Submit.Click()
```

##### П2.2.4.2. Получение сведений о состоянии элемента

Определение состояния кнопки выполняет следующая функция:

***PushButton.GetProperty(property)*** , где *PushButton* – логическое имя кнопки; *property* – название запрашиваемого свойства.

Пример. Строка кода, возвращающая состояние кнопки Submit на данный момент:

```
out_value = Submit.GetProperty("Enabled")
```

где “Enabled” – свойство кнопки, имеющее 2 состояния TRUE и FALSE, *out\_value* – результат операции (true/false)

#### П2.2.5. Функции для CheckBox/RadioButton (Флажок/Переключатель)

Для элементов CheckBox/RadioButton предусмотрены следующие функции:

- выбор параметра;
- отмена выбранного параметра;
- получение информации о состоянии элемента.

#### П2.2.5.1. Выбор параметра

Выбор параметра осуществляется следующими функциями:

***CheckBox.Check()*** , где *CheckBox* – логическое имя кнопки.

***RadioList.Select(sItem)*** , где *RadioList* – логическое имя списка параметров; *sItem* – выбираемый параметр.

Пример 1. Строка кода, выбирающая вариант Meat из списка Toppings = (Pepperony, Meat, Mushrooms):

*Meat.Check()*

Пример 2. Строка кода, выбирающая вариант Large из списка PType = (Large, Medium, Small):

*PType. Select ("Large")*

#### П2.2.5.2. Отмена выбранного параметра

Отмена выбранного параметра осуществляется следующими функциями:

***CheckBox.UnCheck()*** , где *CheckBox* – логическое имя кнопки.

***RadioList.Select (sItem)*** , где *RadioList* – логическое имя списка параметров; *sItem* – выбираемый параметр.

Пример 1. Строка кода, отменяющая выбор варианта Meat из списка Toppings = (Pepperony, Meat, Mushrooms):

*Meat.UnCheck()*

Пример 2. Строка кода, выбирающая вариант Medium из списка PType = (Large, Medium, Small), отменяя тем самым предыдущий выбор Large:

*PType.Select("Medium")*

#### П2.2.5.3. Получение информации о состоянии элемента

Получение информации о состоянии элемента осуществляется следующими функциями:

***CheckBox.GetProperty(property)*** , где *CheckBox* – логическое имя кнопки; *property* – название запрашиваемого свойства.

***RadioList.GetProperty(property)***, где *RadioList* – логическое имя списка параметров; *property* – название запрашиваемого свойства.

Пример 1. Следующая строка кода возвращает текущее состояние параметра Meat из списка Toppings = (Pepperony, Meat, Mushrooms):

*Out\_value=Meat.GetProperty("Value")*

Пример 2. Следующая строка кода возвращает выбранный в данный момент параметр из списка PType = (Large, Medium, Small):

*Out\_value=PType.GetProperty("Value")*

#### П2.2.6. Функции для EditBox (Поле ввода)

Для элемента EditBox предусмотрены следующие функции:

- ввод информации в поле ввода;
- удаление информации;
- получение значения из поля ввода.

#### П2.2.6.1. Ввод информации в поле ввода

Ввод информации осуществляет следующая функция:

***TextField.Set(sValue)***, где *TextField* – логическое имя поля ввода; *sValue* – переменная, хранящая значение поля ввода, или непосредственно поле.

Пример. Ввод текста «Ivanov» в EditBox Name:

```
Name.Set("Ivanov")
```

#### П2.2.6.2. Удаление информации

Удаление информации осуществляет следующая функция:

***TextField.ClearText()***, где *TextField* – логическое имя строки ввода.

Пример. Следующая строка кода удаляет значение из EditBox Name:

```
Name.ClearText()
```

#### П2.2.6.3. Получение значения из поля ввода

Получение информации из поля ввода осуществляет следующая функция:

***TextField.GetText()***, где *TextField* – логическое имя поля ввода.

Пример. Строка кода, возвращающая значение EditBox Name:

```
Out_value=Name.GetText()
```

#### П2.2.7. Функции для Tab (Вкладка)

Функция используется для навигации между различными вкладками, когда скрипт работает с объектами, расположенными на различных вкладках.

***PageList.Select(sValue)***, где *PageLiSt* – логическое имя элемента Tab; *sValue* – переменная, содержащая значение выбираемой вкладки.

#### П2.3. Функции имитации клавиатурного ввода

В языке 4Test, имеющем объектно-ориентированную архитектуру, для имитации клавиатурного ввода реализован метод:

***Object.TypeKeys(keyboard\_input)***, где *Object* – объект, которому передается пользовательский клавиатурный ввод; *keyboard\_input* – строка, описывающая клавиатурный ввод. Обозначение функциональных клавиш, таких как Ctrl, Shift, Alt и т. д., должно начинаться знаком < и оканчиваться знаком >.

Пример 1. Следующая строка кода имитирует нажатие клавиши <F1> и, соответственно, вызов Help:

```
Browser.TypeKeys("<F1>")
```

Пример 2. Следующая строка кода имитирует введение значения Ivan Ivanov в элемент Name класса EditBox. Эта строка кода аналогична использованию метода SetText():

```
Name.TypeKeys("Ivan Ivanov").
```

Следует также отметить наличие методов *PressKeys* и *ReleaseKeys*. Это низкоуровневые методы, призванные решить задачу удержания кнопки в нажатом состоянии в течение определенного времени. При этом могут выполняться другие действия.



***Object.PressKeys(keyboard\_input)***, где **Object** – объект, которому передается пользовательский клавиатурный ввод; **keyboard\_input** – строка, описывающая клавиатурный ввод. Обозначение функциональных клавиш, таких как <Ctrl>, <Shift>, <Alt> и т. д., должно начинаться знаком < и оканчиваться знаком >.

***Object.ReleaseKeys(keyboard\_input)***, где **Object** – объект, которому передается пользовательский клавиатурный ввод; **keyboard\_input** – строка, описывающая клавиатурный ввод. Обозначение функциональных клавиш, таких как <Ctrl>, <Shift>, <Alt> и т. д., должно начинаться знаком < и оканчиваться знаком >.

Пример. Следующий блок кода имитирует введение значения Ivan Ivanov в элемент Name класса EditBox как IVAN IVANOV:

```
Name.PressKeys("<Shift>")  
Name.TypeKeys ("ivan ivanov")  
Name.ReleaseKeys("<Shift>")
```

#### П2.4. Функции имитации действий, выполняемых мышью

4Test предоставляет три метода имитации пользовательского ввода: Click(), DoubleClick() и MultiClick(). Рассмотрим методы Click(), DoubleClick().

***Object.Click([button,x,y])*** , где **Object** – объект, которому передается пользовательский щелчок мыши; **button** – кнопка мыши, используемая для щелчка, может быть 1 (Left), 2 (Right) или 3 (Middle) (необязательный параметр, по умолчанию 1); **x** и **y** – координаты области внутри объекта, в которой будет сделан щелчок (необязательный параметр).

***Object.DoubleClick([button,x,y])*** , где **Object** – объект, которому передается пользовательский щелчок мыши; **button** – кнопка мыши, используемая для щелчка, может быть 1 (Left), 2 (Right) или 3 (Middle) (необязательный параметр, по умолчанию 1); **x** и **y** – координаты области внутри объекта, в которой будет сделан щелчок (необязательный параметр).

Пример. Следующая строка кода имитирует щелчок правой кнопкой мыши на элементе Name класса EditBox:

```
Name.Click (2)
```

#### П2.5. Функции работы с системой

Среди функций работы с системой рассмотрим два метода окна MainWin: Start() и Invoke(). Поскольку 4Test имеет объектно-ориентированную архитектуру, каждое из основных (первых) окон тестируемых или используемых в процессе тестирования программ должно быть определено в файле \*.inc как окно класса MainWin. Метод, описанный далее, является методом именно этого класса.

***MainWin.Start(sCmdLine,sDir,sExtensionReady,nInvokeTimeout)*** , где **sCmdLine** – полный путь и имя программы; **sDir** – путь по умолчанию для запускаемой программы; **sExtensionReady** – функциональный определитель

ожидания загрузки регистра программы; *nInvokeTimeout* – время задержки в секундах.

**Пример.** Следующая строка кода открывает Notepad:

*Notepad.Start ("notepad.exe")*

## П2.6. Функции оповещения о результатах

В 4Test существуют три функции оповещения о результатах: *LogError()*, *LogWarning* и *Print()*. Первые две функции служат для сообщения об ошибке или внештатной ситуации соответственно. Функция *Print()* выводит отладочные сообщения. Рассмотрим эти функции подробнее.

***LogError(error\_msg)*** , где *error\_msg* – строка, описывающая произошедшую ошибку.

***LogWarning (warning\_msg)***, где *warning\_msg* – строка, описывающая произошедшую внештатную ситуацию.

***Print (user\_msg)***, где *user\_msg* – строка, содержащая сообщение пользователя.

Пример 1. Следующая строка кода сообщает о найденной при прохождении скрипта ошибке:

*LogError ("Найдено несоответствие данных")*

Пример 2. Строка кода, сообщающая о том, что при прохождении скрипта не найден объект *Name* класса *EditBox*:

*LogWarning ("Объект класса EditBox не найден")*

Пример 3. Строка кода сообщает текущее значение переменной *J*:

*Print ("Значение переменной j : "+Str(j))*

## П2.7. Функции синхронизации

4Test имеет одну функцию синхронизации *Sleep()*. Эта функция задерживает исполнение скрипта на заданное количество секунд:

***Sleep (seconds)*** , где *seconds* – время задержки в секундах.

## П2.8. Функции проверки существования объекта

В 4Test предоставлен метод проверки существования объекта *Exists()*. Данный метод может быть использован как для окна, так и для любого иного объекта. Рассмотрим этот метод более детально:

***Object.Exists()*** , где *object* – логическое имя объекта/окна.

**Пример.** Следующая строка кода возвращает результат проверки существования окна *Order*:

*result = Order.Exists()*

**Пример.** Строка кода возвращает результат проверки существования объекта *CreditCard*:

*result = CreditCard.Exists()*

## П2.9. Функции работы с файлами

4Test имеет ряд функций работы с файлами: `FileClose()`, `FileOpen()`, `FileReadLine()`, `FileReadValue()`, `FileSetPointer()`, `FileWriteLine()` и `FileWriteValue()`. Рассмотрим эти функции подробнее.

### ***hFile = FileOpen (sPath, fmMode [, fsShare])***

Данная функция открывает файл для работы с ним (запись и/или чтение), где *hFile* – идентификатор (handle) файла типа `HFILE`; *sPath* – полное имя открываемого файла; *fmMode* – свойство открываемого файла, может быть следующим: `FMREAD` – только чтение; `FMWRITE` – только запись (если файл уже существует, его содержимое удаляется); `FMUPDATE` – то же, что и `FMWRITE`, только размер файла сохраняется (но содержимое удаляется); `FMAPPEND` – только запись в конец файла.

### ***FileSetPointer(hFile, iNewValue[, SetMode])***

Данная функция устанавливает позицию чтения/записи в файле, где *hFile* – идентификатор (handle) файла типа `HFILE`; *iNewValue* – новое значение позиции; *setMode* – свойство позиции (необязательный параметр), может быть следующим: `FP_START` – позиция равна *iNewValue* (значение по умолчанию); `FPEND` – позиция равна *iNewValue* + длина файла; `FPRELATIVE` – позиция равна *iNewValue* + нынешняя позиция.

### ***FileReadLine (hFile, sLine)***

Функция считывает одну строку из указанного файла, где *hFile* – идентификатор (handle) файла типа `HFILE`; *sLine* – переменная, хранящая прочитанную строку.

### ***FileReadValue (hFile. aValue)***

Эта функция считывает одну форматированную строку из указанного файла, где *hFile* – идентификатор файла типа `HFILE`; *aValue* – данные определенного формата для записи в файл.

### ***FileWriteLine (hFile, sLine)***

Данная функция записывает одну строку в файл, где *hFile* – идентификатор файла типа `HFILE`; *sLine* – переменная, хранящая выражения для записи в файл.

### ***FileWriteValue (hFile. aValue)***

Функция записывает форматированное значение в файл, где *hFile* – идентификатор файла типа `HFILE`; *aValue* – данные определенного формата, подготовленные для записи в файл.

### ***FileClose (hFile)***

Функция закрывает файл по окончании работы с ним (запись и/или чтение), где *hFile* – идентификатор файла типа `HFILE`.

Пример 1. Следующая строка кода открывает файл `C:\readme.txt`:

*OutputFileHandle = FileOpen ("readme.txt ", FMWRITE)*

Пример 2. Строка кода, устанавливающая позицию чтения/записи на четвертый символ с конца:

*FileSetPointer (OutputFileHandle, -4, FPEND)*

Пример 3. Считывание одной строки из файла `C:\readme.txt`:

*FileReadLine (OutputFileHandle, sLine)*

Пример 4. Считывание одной строки из файла C:\readme.txt (см. описание функции FileWriteValue):

*FileReadValue (OutputFileHandle, Item)*

Пример 5. Строка кода записывает одну строку в файл C:\readme.txt:

*FileWriteLine (OutputFileHandle, " Одна строка ")*

Пример 6. Запись одной строки в файл C:\readme.txt:

*FileWriteValue (OutputFileHandle, "101010101010")*

Пример 7. Следующая строка кода закрывает файл C:\readme.txt, открытый функцией FileOpen:

*FileClose (OutputFileHandle)*

## П2.10. Функции пользователя

4Test предоставляет следующую реализацию функции пользователя:

**[scope][testcase][return-type] function-id ([arguments])statements[return [expression] ]** , где scope – класс функции, public или private; testcase – указатель на то, что данная функция является самостоятельным скриптом; return-type – тип возвращаемого значения; return [expression] – возвращаемое значение.

Определение передаваемого значения имеет следующий синтаксис:

**[pass-mode] data-type identifier [null] [optional]**, где pass-mode – тип передаваемого параметра (in, out или inout: in – параметр, значение которого определено вне данной функции и будет использовано только для чтения; out – параметр, значение которого будет определено в данной функции; inout – параметр, значение которого может быть определено как вне так и внутри данной функции); data-type – тип данных передаваемого значения; null – флаг принятия null-значений; optional – флаг указания необязательности параметра.

Пример. Функция CompareTwoNumbers принимает два численных параметра First и Second, сравнивает два числа и возвращает FALSE, если First меньше Second, и TRUE, если First больше Second:

*Boolean CompareTwoNumbers (INTEGER First, INTEGER Second)*

*If (First < Second)*

*return FALSE*

*else*

*return TRUE*

При активизации функции CompareTwoNumbers(2,1) для указанных параметров возвращаемое значение равно TRUE.

## Практическая часть

### Содержание задания:

Научиться составлять тесты с помощью Silktest. Для заданий №1-10 выполнить детализацию тестов, разработав 4 тест-кейса. Один из тест-кейсов разработать как Data Driven Testcase, организовав чтение исходных данных из Excel-файла. Для разработанных тест-кейсов создать 2 тестплана запуска тест-кейсов. Общие для тест-кейсов шаги инструкций поместить в Recovery-файл.

### Вариант 1:

**Задание №1.** Тест-приложение «Калькулятор». Вид «классический».

Тестирование функции: инженерные расчеты в 8-ой системе счисления.

- 1 Выбрать пункт меню «Вид-Инженерный».
- 2 Проверить доступность кнопок «Ave», «Sta», «7», «8».
- 3 Проверить отмечены ли чекбоксы "Inv" и "Nup".
- 4 Получить значения координат из внешних данных (X и Y).
- 5 Переместить калькулятор по полученным координатам.
- 6 Текущее значение в поле ввода сбросить в 0.
- 7 Протестировать выполнение 2-байтных операций сложения в 8-ой системе счисления.
- 8 Вернуть вид приложения «Вид-Обычный».
- 9 Закрыть окно «Калькулятор».

**Задание №2.** Тест-приложение «Калькулятор». Вид «классический».

Тестирование функции: инженерные расчеты в 8-ой системе счисления.

- 1 Выбрать пункт меню «Вид-Инженерный».
- 2 Проверить доступность кнопок «7», «9», «Backspace», «Sum» .
- 3 Получить значения координат из внешних данных (X и Y).
- 4 Переместить калькулятор по полученным координатам.
- 5 Текущее значение в поле ввода сбросить в 0.
- 6 Протестировать выполнение 1-байтных операций вычитания в 8-ой системе счисления.
- 7 Вернуть вид приложения «Вид-Обычный».
- 8 Закрыть окно «Калькулятор».

### Вариант 2:

**Задание №1.** Тест-приложение «Калькулятор». Вид «классический».

Тестирование функции: инженерные расчеты в 2-ой системе счисления.

- 1 Выбрать пункт меню «Вид-Инженерный».
- 2 Проверить доступность кнопок «2», «3», «4», «5», «0», «1» .
- 3 Получить значения координат из внешних данных (X и Y).
- 4 Переместить калькулятор по полученным координатам.
- 5 Текущее значение в поле ввода сбросить в 0.

- 6 Протестировать выполнение 4-байтных операций умножения в 2-ой системе счисления.
- 7 Вернуть вид приложения «Вид-Обычный».
- 8 Закрыть окно «Калькулятор».

**Задание №2.** Тест-приложение «Калькулятор». Вид «программист».

Тестирование функции: инженерные расчеты в 16-ой системе счисления.

- 1 Выбрать пункт меню «Вид – Инженерный».
- 2 Проверить доступность кнопок «Ave», «Sta», «CE», «C», «D», «E».
- 3 Получить значения координат из внешних данных (X и Y).
- 4 Переместить калькулятор по полученным координатам.
- 5 Текущее значение в поле ввода сбросить в 0.
- 6 Протестировать выполнение 1-байтных операций деления в 16-ой системе счисления.
- 7 Вернуть вид приложения «Вид-Обычный».
- 8 Закрыть главное окно «Калькулятор».

### **Вариант 3:**

**Задание №1.** Тест-приложение «Калькулятор». Вид «программист».

Тестирование функции: инженерные расчеты в 2-ой системе счисления.

- 1 Выбрать пункт меню «Вид-Инженерный».
- 2 Проверить доступность кнопок «2», «3», «4», «5», «0», «1», «MC».
- 3 Получить значения координат из внешних данных (X и Y).
- 4 Переместить калькулятор по полученным координатам.
- 5 Текущее значение в поле ввода сбросить в 0.
- 6 Протестировать выполнение 2-байтных операций деления в 2-ой системе счисления.
- 7 Вернуть вид приложения «Вид-Обычный».
- 8 Закрыть окно «Калькулятор».

**Задание №2.** Тест-приложение «Калькулятор». Вид «программист».

Тестирование функции: статистические расчеты в 2-ой системе счисления.

- 1 Выбрать пункт меню «Вид-Инженерный».
- 2 Проверить доступность кнопок «Sta», «2», «3», «4», «5», «0», «1», «M+».
- 3 Получить значения координат из внешних данных (X и Y).
- 4 Переместить калькулятор по полученным координатам.
- 5 Текущее значение в поле ввода сбросить в 0.
- 6 Протестировать статистические операции «Dat», «Ave», «Sum» для 1-байтных операций в 2-ой системе счисления.
- 7 Вернуть вид приложения «Вид-Обычный».
- 8 Закрыть окно «Калькулятор».

#### **Вариант 4:**

**Задание №1.** Тест-приложение «Калькулятор». Вид «классический».

Тестирование функции: статистические расчеты в 10-ой системе счисления.

- 1 Выбрать пункт меню «Вид-Инженерный».
- 2 Проверить доступность кнопок «Sta», «2», «3», «4», «5», «A», «C».
- 3 Получить значения координат из внешних данных (X и Y).
- 4 Переместить калькулятор по полученным координатам.
- 5 Текущее значение в поле ввода сбросить в 0.
- 6 Протестировать статистические операции «Dat», «S», «Sum» для 10-ой системы счисления.
- 7 Протестировать функции «RET», «LOAD».
- 8 Вернуть вид приложения «Вид-Обычный».
- 9 Закрыть окно «Калькулятор».

**Задание №2.** Тест-приложение «Калькулятор». Вид «классический».

Тестирование функции: статистические расчеты в 16-ой системе счисления.

- 1 Выбрать пункт меню «Вид-Инженерный».
- 2 Проверить доступность кнопок «Sta», «3», «4», «5», «A», «C», «Exp».
- 3 Получить значения координат из внешних данных (X и Y).
- 4 Переместить калькулятор по полученным координатам.
- 5 Текущее значение в поле ввода сбросить в 0.
- 6 Протестировать статистические операции «Dat», «S», «Ave» для 1-байтной операции в 16-ой системе счисления.
- 7 Протестировать функции «CD», «CAD».
- 8 Вернуть вид приложения «Вид-Обычный».
- 9 Закрыть окно «Калькулятор».

#### **Вариант 5:**

**Задание №1.** Тест-приложение «Калькулятор». Вид «классический».

Тестирование функции: работа с памятью в 16-ой системе счисления.

- 1 Выбрать пункт меню «Вид-Инженерный».
- 2 Проверить доступность кнопок «Sta», «Ave», «Dat», «4», «A», «C», «Exp».
- 3 Получить значения координат из внешних данных (X и Y).
- 4 Переместить калькулятор по полученным координатам.
- 5 Текущее значение в поле ввода сбросить в 0.
- 6 Протестировать работу с памятью «MS», «MR», «MC», «M+» для 2-байтных операций в 16-ой системе счисления.
- 7 Закрыть окно «Калькулятор».

**Задание №2.** Тест-приложение «Калькулятор». Вид «классический».

Тестирование функции: работа с памятью в 2-ой системе счисления.

- 1 Выбрать пункт меню «Вид-Инженерный».
- 2 Проверить доступность кнопок «Sta», «Ave», «Cos», «MC», «MR», «MS», «M+».
- 3 Получить значения координат из внешних данных (X и Y).
- 4 Переместить калькулятор по полученным координатам.
- 5 Текущее значение в поле ввода сбросить в 0.
- 6 Протестировать работу с памятью «MS», «MR», «MC», «M+» для 1-байтных операций в 2-ой системе счисления.
- 7 Закрыть окно «Калькулятор».

***Порядок выполнения работы:***

- 1 Изучить теоретическую часть.
- 2 Установить Silktest.
- 3 Автоматизировать тесты с помощью Silktest.
- 4 Составить отчет о проделанной работе.
- 5 Показать работу преподавателю.
- 6 После защиты, выполненного задания, скинуть отчёт преподавателю (в названии указать **фамилию и номер лабораторной**, например, Ivanov14).

***Содержание отчета:***

- 1 Титульный лист.
  - 2 Описание варианта задания.
  - 3 Ход выполнения работы, с пояснением каждого шага и скриншотами.
  - 4 Вывод о проделанной работе.
- Защита работ проводится индивидуально.