

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»

**В. В. Бахтизин, С. С. Куликов, Е. П. Фадеева**

## ***АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ***

*Рекомендовано УМО по образованию в области информатики  
и радиоэлектроники в качестве учебно-методического  
пособия для специальности 1-40 01 01 «Программное обеспечение  
информационных технологий» по дисциплине  
«Надежность программного обеспечения»*

Минск БГУИР 2012

УДК 004.4'22+004.4'6 (076)  
ББК 32.973.202-018.2я7  
Б30

**Р е ц е н з е н т ы:**

кафедра управления информационными ресурсами Академии управления  
при Президенте Республики Беларусь  
(протокол №13 от 21.03.2012г.);

заведующий кафедрой информатики Минского государственного высшего  
радиотехнического колледжа,  
кандидат технических наук, доцент Ю. А. Скудняков

**Бахтизин, В. В.**

Б30 Автоматизация тестирования программного обеспечения. Работа в  
среде SilkTest 8.0 : учеб.-метод. пособие / В. В. Бахтизин, С. С. Куликов,  
Е. П. Фадеева. – Минск : БГУИР, 2012. – 72 с. : ил.  
ISBN 978-985-488-855-2.

Рассмотрена классификация видов тестирования. Сформулированы основные требования к тест-кейсам. Обсуждены вопросы эффективности автоматизации процесса тестирования программного обеспечения. Представлены основные характеристики продуктов-лидеров, предназначенных для автоматизации процесса тестирования. Рассмотрены основные приемы работы при создании тест-кейсов в среде SilkTest 8.0. Дан пример разработки тест-кейса. Предложены контрольные вопросы и приведены задания для выполнения лабораторной работы.

Для специальности 1-40 01 01 «Программное обеспечение информационных технологий» по дисциплине «Надежность программного обеспечения».

**УДК 004.4'22+004.4'6 (076)  
ББК 32.973.202-018.2я7**

**ISBN 978-985-488-855-2**

© Бахтизин В. В., Куликов С. С.,  
Фадеева Е. П., 2012  
© УО «Белорусский государственный  
университет информатики  
и радиоэлектроники», 2012

## СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b> .....	5
<b>1. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ</b> .....	6
1.1. Задача и цели тестирования программного обеспечения.....	6
1.2. Классификация видов тестирования ПО.....	6
1.3. Этапы процесса тестирования ПО .....	10
1.4. Разработка тест-кейсов .....	11
1.5. Контрольные вопросы.....	12
<b>2. АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ</b> .....	14
2.1. Эффективность автоматизации тестирования .....	14
2.2. Технология Record&Playback.....	16
2.3. Язык скриптов.....	17
2.4. Структура автоматизации тестирования .....	17
2.5. Требования к автоматизированному тесту .....	18
2.6. Инструментальные средства автоматизации тестирования.....	18
2.6.1. SilkTest .....	18
2.6.2. HP QuickTest Professional .....	19
2.6.3. Rational Functional Tester.....	19
2.6.4. TestComplete .....	20
2.6.5. Selenium .....	21
2.7. Контрольные вопросы.....	21
<b>3. РАБОТА В СРЕДЕ SilkTest 8.0</b> .....	23
3.1. Начало работы.....	23
3.1.1. Основные понятия .....	23
3.1.2. Создание фреймов .....	23
3.1.3. Структура тест-кейсов.....	24
3.2. Реализация технологии Record&Playback в среде SilkTest .....	24
3.2.1. Создание библиотеки объектов .....	24
3.2.2. Создание тестового скрипта .....	25
3.2.3. Среда запуска скриптов.....	26
3.2.4. Средство распознавания объектов .....	26
3.3. Recovery-система.....	26
3.4. Создание тест-плана (Test Plan) .....	27
3.4.1. Добавление тест-кейса в тест-план .....	27
3.4.2. Запуск тест-плана (полный, выборочный) .....	28
3.5. Использование внешних данных .....	28
3.6. Просмотр результатов исполнения тестовых сценариев.....	33
3.7. Пример разработки теста .....	35
3.7.1. Задание для тестирования .....	35
3.7.2. Создание фрейма приложения.....	37
3.7.3. Настройка Recovery-системы .....	40
3.7.4. Запись и воспроизведение скрипта .....	41
3.7.5. Создание тест-плана .....	49
3.8. Редактирование текста скрипта.....	50
3.9. Задания для выполнения лабораторной работы .....	51
<b>ЛИТЕРАТУРА</b> .....	54

<b>ПРИЛОЖЕНИЕ 1. ОПИСАНИЕ ЯЗЫКА 4Test</b> .....	55
П.1.1. Типы данных .....	55
П.1.1.1. Синтаксис определения переменной .....	55
П.1.1.2. Синтаксис определения константы .....	55
П.1.1.3. Примеры определения и инициализации переменных .....	56
П.1.1.4. Массивы данных .....	56
П.1.2. Операторы и функции языка 4Test .....	56
П.1.2.1. Арифметические операторы .....	56
П.1.2.2. Логические операторы .....	57
П.1.2.3. Операторы сравнения .....	57
П.1.2.4. Условные операторы .....	57
П.1.2.5. Операторы цикла .....	57
П.1.2.6. Функции приведения типа .....	58
П.1.2.7. Функции работы с массивами .....	58
П.1.2.8. Функции работы со строками .....	60
<b>ПРИЛОЖЕНИЕ 2. ЭЛЕМЕНТЫ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ И ИХ ФУНКЦИИ</b> .....	62
П.2.1. Основные элементы графического интерфейса .....	62
П.2.2. Функции элементов графического интерфейса .....	62
П.2.2.1. Функции для Window (окно) .....	62
П.2.2.2. Функции для Menu (меню) .....	63
П.2.2.3. Функции для ListBox/ComboBox (Раскрывающийся список/Поле со списком) .....	63
П.2.2.4. Функции для Button (кнопка) .....	65
П.2.2.5. Функции для CheckBox/RadioButton (Флажок/Переключатель) .....	65
П.2.2.6. Функции для EditBox (Поле ввода) .....	66
П.2.2.7. Функции для Tab (Вкладка) .....	67
П.2.3. Функции имитации клавиатурного ввода .....	67
П.2.4. Функции имитации действий, выполняемых мышью .....	68
П.2.5. Функции работы с системой .....	68
П.2.6. Функции оповещения о результатах .....	69
П.2.7. Функции синхронизации .....	69
П.2.8. Функции проверки существования объекта .....	69
П.2.9. Функции работы с файлами .....	70
П.2.10. Функции пользователя .....	71

## ВВЕДЕНИЕ

Сегодня тестирование стало обязательной частью процесса производства программного обеспечения (далее - ПО).

Компании во всем мире вкладывают все больше средств в процесс тестирования ПО.

С технической точки зрения тестирование заключается в выполнении приложения на некотором множестве исходных данных и сверке получаемых результатов с заранее известными (эталонными) с целью установить соответствие различных свойств и характеристик приложения заданным требованиям.

Как один из основных этапов процесса разработки программного продукта (дизайн приложения – разработка кода – тестирование), тестирование характеризуется большим вкладом в суммарную трудоемкость разработки продукта. Известна оценка распределения трудоемкости между этапами создания программного продукта: 40 % – 20 % – 40 %. Из чего следует, что наибольший эффект в снижении трудоемкости может быть получен прежде всего на этапах разработки дизайна приложения и тестирования. Один из путей снижения трудоемкости процесса разработки – это его автоматизация.

Успешная автоматизация тестирования ПО дает следующие преимущества:

- сокращение времени исполнения тестов;
- надежность, поскольку устраняется человеческий фактор;
- мощность, например человек никогда не сможет вручную добавить в БД 100 миллиардов записей, составленных из 500 полей каждая;
- сбор статистической информации о работе приложения и представление ее в удобной для восприятия человеком форме;
- средства автоматизации тестирования в сложных ошибочных ситуациях способны выполнять «низкоуровневые действия», сложные для человека.

Сегодня среди средств автоматизации тестирования программного обеспечения лидируют следующие: SilkTest, HP QuickTest Professional (QuickTestPro или QTP), Rational Functional Tester, SmartBear TestComplete, Selenium.

В учебно-методическом пособии описываются приемы работы в одной из последних версий SilkTest.

# 1. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## 1.1. Задача и цели тестирования программного обеспечения

Тестирование программного обеспечения (software testing) – это процесс анализа или эксплуатации программного обеспечения с целью выявления в нем дефектов, ошибок (defect, bug).

Задача тестирования ПО – выявление дефектов в программном обеспечении до того, как эти дефекты будут обнаружены заказчиком или конечным пользователем.

Тестированию подлежат:

- 1) программа при ее непосредственном запуске и исполнении (software);
- 2) код программы без запуска и исполнения (code);
- 3) прототип программного продукта (product prototype);
- 4) проектная документация (project documentation):
  - требования к программному продукту (product requirements);
  - функциональные спецификации к программному продукту (functional specifications);
  - документы, описывающие архитектуру (product architecture), дизайн (product design);
  - план проекта (project plan) и тестовый план (test plan);
  - тестовые сценарии (test case);
- 5) сопроводительная документация и документация для пользователей:
  - интерактивная помощь (on-line help);
  - руководства по установке (installation guide) и использованию программного продукта (user manual).

Необходимость тестирования любого из указанных рабочих продуктов определяется проектом и условиями его разработки.

Цель тестирования ПО – это выявление и устранение ошибок в тестируемом объекте. Однако полностью протестировать даже несложную программу невозможно. Обычная практика тестирования такова: программное средство считается пригодным к выпуску, если в нем устранены все обнаруженные критические ошибки и 85 % некритических ошибок [8].

## 1.2. Классификация видов тестирования ПО

Тестирование ПО можно классифицировать по следующим признакам [1–8,10,11]:

- по степени использования тестируемого ПО;
- по знанию системы;
- по объекту тестирования;
- по степени изолированности компонентов;

- по признаку позитивности сценариев;
- по времени проведения тестирования;
- по степени подготовленности к тестированию;
- по уровню тестирования;
- по степени автоматизации.

По степени использования тестируемого ПО выделяют:

- статическое тестирование (static testing);
- динамическое тестирование (dynamic testing).

Статическое тестирование – это процесс анализа самой разработки программного обеспечения, т. е. тестирование без запуска программы. Статическое тестирование предусматривает проверку программного кода, требований к программному продукту, функциональной спецификации, архитектуры, дизайна и т. д.

Динамическое тестирование – это тестовая деятельность, предусматривающая эксплуатацию (запуск) программного продукта. Динамическое тестирование предполагает запуск программы, выполнение всех ее функциональных модулей и сравнение фактического ее поведения с ожидаемым.

По знанию системы выделяют:

- тестирование «черного ящика» (black box testing);
- тестирование «белого ящика» (white box testing);
- тестирование «серого ящика» (grey box testing).

При тестировании по методу «черного ящика» идеи для тестирования формируются путем предположений о сценариях, которые будут реализовываться и применяться пользователями. Поэтому метод «черного ящика» также называют «поведенческим тестированием». Примером такого тестирования является функциональное тестирование. Тестировщик запускает приложение на выполнение и тестирует его функциональность, используя пользовательский интерфейс для ввода входных и получения выходных данных.

«Белый ящик» также известен под именами «стеклянный ящик» (glass/clear box), открытый ящик (open box) и даже никакой ящик (no box). При тестировании по методу «белого ящика» тестировщик основывает идеи для тестирования на знании устройства и внутренней логики тестируемой программы, а не на образцах поведения пользователей. В реальной жизни тестирование по методу «белый ящик» проводится либо самими программистами, написавшими код, либо их коллегами с программистской квалификацией. Юнит-тестирование (unit-testing) – это часть тестирования по методу «белого ящика».

«Серый ящик» – метод, сочетающий элементы «белого ящика» и «черного ящика». С одной стороны, тестирование, ориентированное на пользователя, а значит, используются сценарии поведения пользователя («черный ящик»), с другой, информированное тестирование, т. е. тестирование со знанием, как устроена хотя бы часть анализируемой программы («белый ящик»).

По объекту тестирования различают:

- функциональное тестирование (functional testing);
- тестирование интерфейса пользователя (UI testing);

- тестирование интернационализации и локализации (internationalisation testing, localization testing);
- тестирование производительности (load/stress/performance testing);
- тестирование безопасности (security testing);
- тестирование удобства использования (usability testing);
- тестирование совместимости (compatibility testing).

Функциональное тестирование проводится с целью выявления ошибок в функционировании программы путем сопоставления фактического (actual) и ожидаемого (expected) результатов. Для получения фактического результата программу надо выполнить. Основным источником ожидаемого результата являются спецификации функциональных требований (requirements) к программному продукту.

Тестирование интерфейса пользователя – это тестирование, при котором проверяются элементы интерфейса пользователя.

Тестирование интернационализации – вид тестирования, при котором проверяется готовность приложения к работе с различными языковыми настройками, в частности способность корректно отображать шрифты, пункты меню, производить поиск, сортировку, способность приложения обрабатывать файлы, поименованные на различных языках.

Локализационное тестирование проверяет, насколько корректно продукт адаптирован к работе на том или ином языке: все ли переведено и переведено правильно, не нарушилась ли логика построения интерфейса и обработки данных и т. д.

Тестирование производительности проводится с целью определения скорости работы системы или ее частей при воздействии определенной нагрузки. Выделяют нагрузочное и стрессовое тестирование. Нагрузочное тестирование проверяет способность приложения работать при запланированной нагрузке. Стрессовое тестирование проверяет поведение приложения в условиях, выходящих за рамки, оговоренные в эксплуатационной документации. Целью такого тестирования является обнаружение слабого места в системе, включая код, базу данных, работу с аппаратными компонентами системы и т. д.

Тестирование безопасности производится для оценки уязвимости программного обеспечения к различным вредоносным воздействиям на уровне приложения, операционной системы, сети и т. д.

Тестирование удобства использования выполняется с целью определения, насколько органично используется пользовательский интерфейс целевыми пользователями, т. е. проверяется интуитивность интерфейса.

Тестирование совместимости – это вид тестирования, основной целью которого является проверка качественной работы разрабатываемого программного средства с другим ПО (операционными системами, браузерами и т. д.). Тестирование с разными браузерами называется кросс-браузерным тестированием (cross-browser testing). Тестирование с разными операционными системами называется кросс-платформенным тестированием (cross-platform testing).

По степени изолированности тестируемых компонентов выделяют:



- компонентное тестирование (component testing);
- интеграционное тестирование (integration testing);
- системное тестирование (system/end-to-end testing).

Компонентное тестирование – это тестирование логических компонентов программного продукта.

Интеграционное тестирование – это тестирование взаимодействия двух или более компонентов.

Системное тестирование – это проверка всей системы целиком (в сборе).

По критерию «позитивности» сценариев различают тестирование:

- позитивное (positive testing);
- негативное (negative testing).

Позитивное тестирование проводится по сценариям, предполагающим нормальную, «правильную» работу системы в заведомо корректных условиях.

Негативное тестирование использует сценарии, проверяющие ситуации, связанные с потенциальными дефектами в системе. Например, проверка работы программы при вводе недействительных данных.

При наличии позитивных и негативных сценариев тестирования приоритет исполнения имеют позитивные сценарии.

По времени проведения тестирования выделяют:

тестирование до передачи пользователю или альфа-тестирование (alpha testing);

тестирование после передачи пользователю или бета-тестирование (beta testing).

Обычно в качестве альфа-тестировщиков выступают сотрудники компании: тестировщики, программисты, системные администраторы и т. д. Юнит-тестирование является видом альфа-тестирования.

Бета-тестировщики – это ограниченная группа пользователей из целевой аудитории. Они работают с новой системой до того, как она станет доступна всем пользователям.

По уровню тестирования выделяют:

- тест приемки (smoke test, sanity test или confidence test);
- тестирование новой функциональности (new feature testing);
- регрессионное тестирование (regression testing);
- тест критического пути (critical path test);
- расширенный тест (extended test);
- тест сдачи (acceptance or certification test).

Приемочный тест – это самый первый и короткий тест, проверяющий работу основной функциональности программного продукта. По результатам этого теста принимается решение о целесообразности дальнейшего тестирования. Если программа не прошла приемочный тест, она отправляется на доработку к программистам.

При тестировании новой функциональности акцент делается на тестировании новых функций, появившихся в конкретном выпуске (build) программного продукта.

Регрессионное тестирование – повторное выполнение тестов для проверки того, что изменения, внесенные в программу в результате разработки новых или изменения существующих функций, устранения ошибок, не повлияли на функциональность, которая не изменялась.

Во время теста критического пути проверяется основная функциональность программного продукта, критичная для конечного пользователя при стандартном использовании продукта.

Расширенный тест – это углубленный тест, при котором проверяется нестандартное использование программного продукта, т. к. программа должна корректно реагировать на любые действия пользователя и работать надежно в любой ситуации. Это особенно актуально для программ медицинской, финансовой, военной, космической направленности.

По степени автоматизированности тестирования различают:

- ручное тестирование (manual testing);
- автоматизированное тестирование (automated testing);
- смешанное/полуавтоматизированное тестирование (semi automated testing).

Ручное тестирование – это исполнение тестов без помощи каких-либо программ, автоматизирующих работу тестировщика. Ручное тестирование – очень трудоемкий процесс.

Автоматизированное тестирование – вид тестирования, при котором используются специальные программные средства для выполнения тестов, сверки полученных фактических результатов с ожидаемыми значениями, установки предусловий для запуска тестов, а также для других функций, включая генерацию отчетов.

Смешанное/полуавтоматическое тестирование – это сочетание ручного подхода с автоматизированным.

### **1.3. Этапы процесса тестирования ПО**

Процесс тестирования ПО включает этапы, представленные на рис. 1.1 [8,10].

Этап планирования тестовых испытаний – это разработка тестовой документации, включающей описание типов тестов, тестируемых составляющих, требуемых ресурсов.

Этап проектирования тестов предполагает проведение работ по созданию и поддержке тестовых сценариев, разработке тестовых спецификаций.

Реализация тестов – это практический поиск дефектов.

На этапе разработки тестов тестировщик выполняет также разработку вспомогательных программ (tools), облегчающих исполнение тестирования путем его автоматизации.



Рис. 1.1

## 1.4. Разработка тест-кейсов

Ядро тестовой документации составляют тест-кейсы [8,10].

Тест-кейс – это набор входных данных, условий выполнения и ожидаемых результатов, разработанный с целью проверки некоторого свойства или поведения программного средства. Дословный перевод «test case» – тестовый случай или тестовая оболочка.

На основе тест-кейсов составляются тестовые комплекты (test suite).

Тестовый комплект обычно формируют из тест-кейсов, проверяющих определенную часть проекта.

Основные элементы тест-кейса:

- ожидаемый результат;
- инструкция исполнителю для получения фактического результата;
- входные данные для реализации шагов инструкции.

Инструкция представляет собой набор шагов (steps) для исполнения теста.

Количество ожидаемых результатов может быть более одного. При наличии нескольких ожидаемых результатов исполнение тест-кейса будет иметь положительный исход, когда все ожидаемые результаты будут равны всем фактическим результатам.

При документировании тест-кейса ему присваивается уникальный идентификационный номер и приоритет. Установка приоритета необходима для оцен-

ки важности тест-кейса. Обычно используют шкалу от 1 до 4. Приоритет 1 – наивысший приоритет. Учет приоритетов тест-кейсов важен при регрессионном тестировании.

Описание тест-кейса также включает:

- краткое заглавие теста (идея теста), т. е. описание конкретного предмета или ситуации, проверяемой тест-кейсом;
- ссылку на связанное с тестом требование;
- ссылку на модуль и подмодуль программного средства, к которым относится тест;
- подготовительную часть, содержащую данные, упрощающие исполнение и поддержку тест-кейса (вплоть до комментариев в помощь тестирующему);
- историю редактирования (атрибуты истории редактирования: тест-кейс создан «кем, дата», тест-кейс изменен «кем, дата», «что, зачем и почему было изменено»);
- автора теста, время последнего выполнения теста, последний полученный актуальный результат, связанный с тестом отчет о дефекте (bug report).

При разработке тест-кейса руководствуются следующими правилами:

- отсутствие ссылок на другие тест-кейсы;
- независимость тест-кейсов, т. е. возможность их исполнения в любом порядке;
- четкость шагов при написании инструкции;
- четко сформулированная тестовая идея.

Тест-кейс, исполнение которого завершено, дает один из двух исходов:

- положительный исход (pass), если фактический результат равен ожидаемому результату;
- отрицательный исход (fail), если фактический результат не равен ожидаемому.

Наиболее значимым с точки зрения тестирования является исход fail, поскольку в ПО обнаружен дефект и начинается работа по его устранению.

## 1.5. Контрольные вопросы

1. Что понимают под тестированием программного обеспечения?
2. Какова основная цель тестирования?
3. Что такое тест-кейс, тестовый комплект и тестовый план?
4. Перечислите атрибуты тест-кейса и причину полезности каждого из них.
5. Предложите тест-кейс с одной идеей и двумя ожидаемыми результатами.
6. Когда при наличии нескольких ожидаемых результатов исполнение тест-кейса будет иметь положительный исход?
7. Для чего ведется история изменения тест-кейсов?
8. Для чего нужна приоритезация тест-кейсов? Каким тест-кейсам присваивается приоритет 1?

10. Что такое Smoke Test?
11. Что такое Extended Test?
12. Что такое Critical Path Test?
13. Что происходит, если тест приемки не пройден?
14. Что проверяется во время теста критического пути?
15. Что проверяется во время расширенного теста?
16. Какими правилами руководствуются при разработке тест-кейсов?
17. Предложите примеры правильно и неправильно написанных шагов тестовой инструкции.
18. Почему тест, не пройденный, считается успешным, а тест, пройденный, потерей времени?
19. В чем заключается процедура фиксирования дефектов?
20. В чем заключается жизненный цикл дефектов?
21. Какие системы управления жизненным циклом дефектов (Bug Tracking System) вы знаете?
22. Чем различаются динамическое и статическое тестирование?
23. Кто может выступать альфа-тестировщиком?
24. Кто может выступать бета-тестировщиком?
25. Назовите атрибуты истории редактирования тест-кейса.
26. Назовите статусы состояний тест-кейса.
27. Как меняется статус тест-кейса, когда в тестируемом ПО исключается функциональность, которую он проверял?

## 2. АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ ПО

### 2.1. Эффективность автоматизации тестирования

Автоматизация тестирования (test automation) – набор техник, подходов и инструментальных средств, позволяющих исключить человека из выполнения некоторых задач в процессе тестирования [2,8–10].

Инструментальное средство автоматизированного тестирования (test automation tool) – программа (или набор программ), позволяющая создавать, редактировать, отлаживать и выполнять автоматизированные тесты, а также собирать статистику их выполнения.

В табл. 2.1 представлены работы, составляющие суть процесса тестирования, и способы их возможного выполнения [10]. Как видно, большинство работ в области тестирования могут быть выполнены только ручным способом, и лишь при выполнении некоторых из них могут быть использованы подходы, методы и инструментальные средства автоматизации.

Таблица 2.1

Виды работ по тестированию

Наименование работы	Способ выполнения
Анализ документации и спецификаций на наличие разнообразных ошибок и недоработок	Ручной
Анализ исходного кода на наличие логических ошибок	Ручной
Разработка тестов	Ручной
Создание алгоритмов формирования тестовых данных	Ручной
Первый запуск тестов	Ручной
Экспертная оценка результатов выполнения тестов	Ручной
Написание отчетов об ошибках	Ручной
Отладка и обновление тестов	Ручной
Документирование процесса тестирования	Ручной
Выбор тестов, которые необходимо выполнить	Ручной
Построение матрицы покрытия требований тестами	Ручной или автоматизированный
Выполнение тестов	Ручной или автоматизированный
Запись результатов выполнения тестов	Ручной или автоматизированный
Статистическая оценка результатов выполнения тестов	Ручной или автоматизированный
Измерение числовых характеристик выполнения тестов (производительность)	Ручной или автоматизированный

Опыт тестирования позволил выделить несколько областей, в которых автоматизация тестирования оказывается наиболее оправданной:

- выполнение smoke test для крупных систем, где приходится выполнять большое количество примитивных трудоемких задач;
- регрессионное тестирование, в силу того, что одни и те же тесты выполняются многократно;
- конфигурационное тестирование для проверки работоспособности приложения при тех или иных настройках (например, у приложения 100 параметров в файле конфигураций, каждый параметр имеет 100 значений, в результате получаем 10 000 вариантов конфигурационных файлов);
- распределенное тестирование, эмулирующее работу большого количества серверных и клиентских компонентов;
- часто повторяющиеся простые тесты (например аутентификация пользователя, регистрация пользователя);
- длинные, утомительные для человека тесты (например заполнение большого количества форм);
- тестирование web-ориентированных приложений для проверки «битых ссылок», соответствия HTML/CSS-кода стандартам и его совместимости с различными браузерами;
- нагрузочное, стрессовое тестирование и тестирование производительности. Например, необходимость проэмулировать работу нескольких десятков тысяч пользователей с одновременным сбором статистики по работе каждого из них и реакции приложения.

Успешная автоматизация дает следующие преимущества:

- скорость выполнения тестирования;
- надежность, поскольку устраняется человеческий фактор;
- мощность, т. к. человек никогда не сможет вручную добавить в базу данных 100 миллиардов записей, составленных из 500 полей каждая;
- сбор статистической информации и представление ее в удобной для человека форме;
- средства автоматизации тестирования в сложных ошибочных ситуациях способны выполнять «низкоуровневые действия», сложные для человека.

В то же время автоматизация тестирования имеет недостатки:

- на разработку автоматизированных тестов уходит в 5–10 раз больше времени, чем на создание и однократное выполнение аналогичных ручных тестов;
- высокая стоимость создания автоматизированных тестов, включающая необходимость приобретения специального программного обеспечения;
- обучение специалистов;
- автоматизированное регрессионное тестирование реже обнаруживает новые ошибки, т. к. тесты часто проходят строго по одному и тому же пути;

– непродуманность вопросов поддержки автоматизированных тестов приводит к тому, что в случае изменений в приложении автоматизированные тесты требуют серьезной переработки.

Каждый проект требует своего подхода к автоматизации.

Приступая к автоматизации, следует учесть затраты времени на ручное выполнение тестов, количество повторений тестов, затраты времени на отладку и обновление автоматизированных тестов, затраты времени на выполнение автоматизированных тестов, затраты времени на поддержку автоматизированных тестов и т. д.

Приведем простой пример подсчета эффективности автоматизации тестов, сравнив время на подготовку и создание автоматизированного и ручного тестов. В общем случае специфика проверяемой программы роли не играет и данное сравнение применимо практически к любой ситуации. Если среднее время на создание теста при ручном тестировании составляет 6 мин ( $T_{cm} = \text{TestCreation, manual}$ ), то при автоматизированном – 30 мин ( $T_{ca} = \text{TestCreation, automated}$ ). Среднее время на прогон теста вручную – 10 мин ( $T_{em} = \text{TestExecution, manual}$ ), автоматически – 1 мин ( $T_{ea} = \text{TestExecution, automated}$ ). Сравнив величины  $T_{cm} + T_{em} \times N$  и  $T_{ca} + T_{ea} \times N$ , где  $N$  – количество итераций теста, можно увидеть, что только при повторении теста более 10 раз автоматизация становится уместной.

## 2.2. Технология Record&Playback

Для ускорения процесса разработки автоматизированных тестов большинство средств автоматизации процессов тестирования используют технологию Record&Playback (Запись и воспроизведение) [9].

Суть технологии Record&Playback заключается в том, что средство автоматизации тестирования позволяет выполнить с тестируемым приложением некоторый набор действий, которые будут записаны на специальном языке программирования и представлены в виде скрипта.

Скрипт – это логически законченная часть кода, сохраненная в отдельном файле и являющаяся программной реализацией определенного тест-кейса.

Для реализации технологии Record&Playback средство автоматизации тестирования должно иметь средство распознавания объектов, библиотеку объектов и среду выполнения.

Средство распознавания объекта – это утилита распознавания класса объекта и его свойств.

Библиотека объектов – это место, где полученная информация будет храниться в специальном формате.

Среда выполнения – это оболочка, которая позволяет исполнять и при необходимости модифицировать скрипты автоматизации.

Преимущества технологии Record&Playback:

- процесс создания исходного «скелета» тестов значительно ускоряется;



- средство автоматизации само собирает необходимую техническую информацию о тестируемом приложении.

Недостатки технологии Record&Playback:

- записанные таким образом тесты «жестко закодированы» («hard-coded»);

- средство автоматизации записывает все, что происходит в момент записи теста (лишнее затем приходится удалять вручную);

- если в процессе записи тестировщик выполнил неверное действие, часть теста или даже весь тест придется перезаписать;

- если приложение достаточно сильно изменилось, тест придется полностью перезаписать.

Данная технология хороша в качестве помощника, но не позволяет создавать сложные тесты.

### 2.3. Язык скриптов

Разработка сложных автоматизированных тестов выполняется с использования скриптовых языков.

Язык скриптов – это высокоуровневый язык программирования, оперирующий терминами и объектами среды, в которой он исполняется. Будучи скомпилированным в машинный код, программный скрипт заставляет тестируемую программу выполнять определенные действия, заменяя пользователя.

Языки скриптов в зависимости от архитектуры, выбранной поставщиком программного продукта автоматизации процессов тестирования, могут быть самыми разными. Например, средство автоматизации тестирования SilkTest для разработки скриптов использует язык 4Test, имеющий объектно-ориентированную архитектуру.

### 2.4. Структура автоматизации тестирования

Программная часть автоматизации тестирования состоит из 3 компонентов:

- библиотеки функций (function library). Представляет собой набор пользовательских функций, используемых скриптами. Основным критерием при ее создании является дублирующаяся функциональность. Пример: функция авторизации пользователя, которая может вызываться многими тестами;

- библиотеки объектов (object repository). Представляет собой описание всех графических объектов программного продукта. Прежде чем средство автоматизации процессов тестирования сможет работать с каким-либо элементом графического интерфейса пользователя, этот элемент должен быть описан в соответствующем формате и сохранен в специальном файле;

- библиотеки скриптов (script library). Представляет собой набор скриптов, выполняющих задачи автоматического тестирования программного про-

дукта. Эта часть автоматизации является основным элементом всего процесса тестирования.

Помимо программной части для автоматизации процессов тестирования важны также наборы данных, включающие ожидаемые результаты, и данные, вводимые при исполнении тестов.

## 2.5. Требования к автоматизированному тесту

Автоматизация тестирования начинается с ручного тестирования. Каждый скрипт базируется на ручном тесте, представленном с должным уровнем детализации. Автоматизированный тест подобен тесту, разработанному для ручного тестирования. По аналогии с ручным сценарием для автоматизированного сценария необходимо выделить последовательность шагов, выполнение которых приведет к значимому для пользователя, системы или автоматизируемого процесса результату, который, в свою очередь, может быть зафиксирован и оценен.

Однако есть ряд проблем, характерных только для автоматизированных тестов:

- необходимость синхронизации работы средства автоматизации и тестируемого приложения по времени;
- ожидаемый результат в автоматизированных тестах должен быть описан предельно четко с указанием конкретных признаков его корректности;
- тесты следует проектировать в виде тестов, управляемых данными (Data Driven Test), для работы с внешними данными, принимающими широкий диапазон форм представления и значений; использование технологии Data Driven Testing (DDT) также упрощает дальнейшую поддержку и модификацию тестов;
- тест может быть автоматизирован с использованием различных инструментальных средств, поэтому его следует описывать, избегая специфических для того или иного инструментального средства решений.

## 2.6. Инструментальные средства автоматизации тестирования

### 2.6.1. *SilkTest*

Поставщик: Segue Software.

*SilkTest* – инструмент автоматизированного тестирования прикладных программ через графический интерфейс пользователя [14]. Программа предназначена для функционального, регрессионного, кросс-платформенного и локализационного тестирования для широкого диапазона технологий разработки приложений.

Основные возможности:

- использование технологии Data Driven Testing;
- функция Code Completion для быстрой настройки тестов и разработки инфраструктуры автоматизации, интеграция с коллективными коммуникационными средствами Rational;

– выполнение проверок баз данных с помощью стандартного доступа через ODBC, что гарантирует точность выполнения сложных транзакций.

Поддерживаемые технологии: AJAX/Flex/ Web 2.0, Java, .NET клиент/сервер. Поддержка различных браузеров без модификации скриптов. Поддержка Eclipse, Java и JUnit

Поддерживаемые операционные системы: Microsoft Windows.

Язык скриптов: 4Test.

Тестируемые приложения: Web, Java, .NET и клиент-серверные приложения.

### 2.6.2. HP QuickTest Professional

Поставщик: Hewlett Packard (HP).

Quicktest Professional (QuickTestPro или QTP) – основной инструмент автоматизации функционального тестирования [12]. QTP позволяет автоматизировать функциональные и регрессионные тесты путем записи действий пользователя при работе с тестируемым приложением и дальнейшего исполнения записанных действий с целью проверки работоспособности ПО. Записанные действия сохраняются в виде скриптов. Скрипты отображаются как VBScript (Expert View) или как визуальные последовательные шаги с действиями (Keyword View). Каждый шаг скрипта может быть отредактирован и дополнен точками проверки (CheckPoint) для сравнения ожидаемого результата с фактическим.

Основные возможности:

- обработка исключительных ситуаций (Exception Handling);
- формирование данных в таблицы с последующим использованием Data Driven Testing;
- работа со сложными UI-объектами;
- расширяемость за счет дополнительных модулей;
- формирование отчетов с результатами выполнения тестирования.

Поддерживаемые технологии: Web, Java, .Net, WPF, SAP, Oracle, Siebel, PeopleSoft, Delphi, Power Builder, Stingray 1, Terminal Emulator, Flex, Mainframe terminal emulator.

Поддерживаемые операционные системы: Microsoft Windows.

Язык скриптов: VBScript.

Тестируемые приложения: Web, Java, .NET и клиент-серверные приложения.

### 2.6.3. Rational Functional Tester

Поставщик: IBM Rational Software.

Rational Functional Tester – инструмент автоматизированного тестирования, позволяющий выполнять функциональное тестирование, регрессионное тестирование, тестирование пользовательского интерфейса и тестирование, управляемое данными [13].

Основные возможности:

- интеграция с коллективными коммуникационными средствами Rational;

- интеграция в среду Eclipse, WebSphere Studio и Rational XDE Developer;
- возможность автоматизировать тестирование, устойчивое к частым изменениям пользовательского интерфейса приложений, благодаря технологии ScriptAssure;
- проверка динамических данных с использованием различных мастеров, точек проверки и шаблонов регулярных выражений;
- наличие автоматизированного мастера для создания Data Driven Test;
- выбор языка сценариев для разработки и настройки тестов: Java в среде Eclipse или Microsoft Visual Basic, .NET в среде Visual Studio .NET;
- поддержка пользовательских элементов управления благодаря прокси-объекту SDK (Java/.Net);
- поддержка функционального тестирования сред приложений Oracle ERP посредством поставляемых расширений.

Поддерживаемые технологии: .Net, Java, Siebel, SAP, терминальные приложения, приложения PowerBuilder, AJAX, Adobe Flex, Dojo Toolkit, GEF, документы Adobe PDF, приложения zSeries, iSeries и pSeries.

Поддерживаемые операционные системы: Microsoft Windows, Linux.

Язык скриптов: VBScript, Java.

Тестируемые приложения: Java-приложения, Web-приложения.

#### 2.6.4. TestComplete

Поставщик: SmartBear Software.

TestComplete – полнофункциональная система для автоматизации тестирования приложений [15]. С помощью TestComplete можно выполнять функциональное, регрессионное, распределенное тестирование, а также тестирование работоспособности HTTP на проектном уровне.

Основные возможности:

- встроенный редактор кода, помогающий тестировщикам писать скрипты вручную;
- режим Record/Playback с записью только ключевых действий;
- отладчик скриптов для построчного выполнения теста;
- распознавание объектов приложений, написанных на Delphi, C++Builder, .Net, WPF, Java и Visual Basic;
- использование шаблонов основных баг-трекговых систем: Microsoft Visual Studio 2005, 2008, 2010 Team System, BugZilla и AutomatedQA AQdevTeam;
- компилятор TestComplete независим от языка программирования и основан на открытом API, COM интерфейсе;
- возможность автоматической фиксации снимков экрана во время записи и выполнения тестов для последующего сравнения ожидаемого и фактического результатов.

Поддерживаемые технологии: 32-битные и 64-битные Windows приложения, .NET (C#, VB.NET, JScript.NET, VCL.NET, C#Builder, Python .NET, Perl

.NET, WPF, Java), Sybase PowerBuilder, Microsoft FoxPro, Microsoft Access, Microsoft InfoPath, Web Browsers (Internet Explorer, Firefox, Netscape Navigator), Visual C++, Visual Basic, Delphi, C++Builder, Flex, Flash, Silverlight, Windows Mobile Applications.

Поддерживаемые операционные системы: Microsoft Windows 2000, XP, Server 2003, Server 2008, Vista, Windows 7.

Язык скриптов: VBScript, JScript, DelphiScript, C++Script, C#Script.

Тестируемые приложения: Web, Java, .NET и клиент-серверные приложения.

### 2.6.5. Selenium

Selenium – инструмент для тестирования Web-приложений [16]. Selenium – JavaScript приложение для управления браузером и выполнения необходимых действий и проверок над приложением. Selenium поддерживается браузерами Microsoft Internet Explorer, Google Chrome, Mozilla Suite и Mozilla Firefox для Microsoft Windows, Linux, Apple Macintosh.

В рамках проекта Selenium выпускаются инструменты Selenium IDE (IDE – Integrated Development Environment) и Selenium Web Driver. Это интегрированные среды с открытым кодом для разработки и выполнения скриптов.

Selenium IDE (самый простой из продуктов семейства Selenium) можно скачать из Internet по адресу: <http://seleniumhq.org/projects/ide/> и использовать бесплатно.

Selenium IDE предоставляет утилиту для записи и последующего воспроизведения тестов. Тестовые скрипты могут быть автоматически записаны и изменены вручную. Selenium Remote Control – сервер, написанный на Java, воспринимает команды для браузера через HTTP-протокол. Selenium Remote Control делает возможным написание тестов для Web-приложений на любом удобном языке программирования, что позволяет интегрировать Selenium в качестве фреймворка для Unit-тестирования. Selenium Web Driver – средство управления браузерами не через JavaScript, а через API-вызовы браузера.

Поддерживаемые операционные системы: Windows, Linux и Macintosh.

Язык скриптов: предметно-ориентированный язык программирования DSL (Domain Specific Language) для написания тестов и преобразования в код любого из языков программирования: C#, Java, Groovy, Perl, PHP, Python и Ruby.

Тестируемые приложения: Web-приложения.

## 2.7. Контрольные вопросы

1. Назовите задачи тестирования, для решения которых автоматизация тестирования является наиболее оправданной.
2. Назовите преимущества, которые дает автоматизация тестирования.
3. Назовите недостатки автоматизации тестирования.

4. Назовите виды работ по тестированию ПО, где могут быть разработаны и использованы средства автоматизации тестирования.
5. Что такое технология Data Driven Testing?
6. В чем заключается удобство поддержки тест-кейса, разработанного с использованием технологии Data Driven Testing?
7. Какие сегодня продукты средств автоматизации тестирования и фирмы лидируют в области разработки средств автоматизации тестирования?
8. Чем разработка ручного тест-кейса отличается от разработки автоматизированного тест-кейса?
9. Что представляет собой программная часть автоматизации тестирования?
10. В чем суть технологии Record&Playback?
11. Какие тесты могут быть разработаны с использованием технологии Record&Playback?
12. Что такое скрипт?
13. Для чего нужны языки скриптов?
14. Назовите источники фактического результата при тестировании ПО.
15. Назовите основные элементы тест-кейса.
16. Почему важна независимая организация тест-кейсов?
17. Какие рабочие продукты при разработке ПО подлежат тестированию?
18. Какими психологическими качествами и техническими навыками должен обладать хороший тестировщик?
19. Назовите этапы процесса тестирования.
20. Дайте классификацию видов тестирования.
21. Как выполняется динамическое тестирование?
22. Почему тестирование по методу «черного ящика» называют поведенческим?
23. Назовите источники информации о функциональностях ПО.
24. Что такое регрессионное тестирование?
25. Назовите ситуации, при которых проводится регрессионное тестирование.
26. Предложите формулу для расчета эффективности автоматизации тестирования.

## 3. РАБОТА В СРЕДЕ SILKTEST 8.0

### 3.1. Начало работы

#### 3.1.1. Основные понятия

Основные понятия SilkTest: фрейм и тест-кейс [8]. 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.

#### 3.1.2. Создание фреймов

Существуют два способа записи нового окна:

- 1) меню File-New-Test Frame (рис. 3.1);

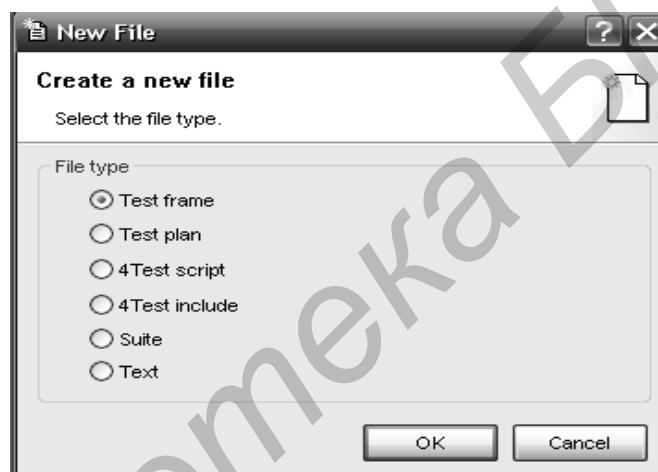


Рис. 3.1

- 2) меню Record-Window Declarations (рис. 3.2).

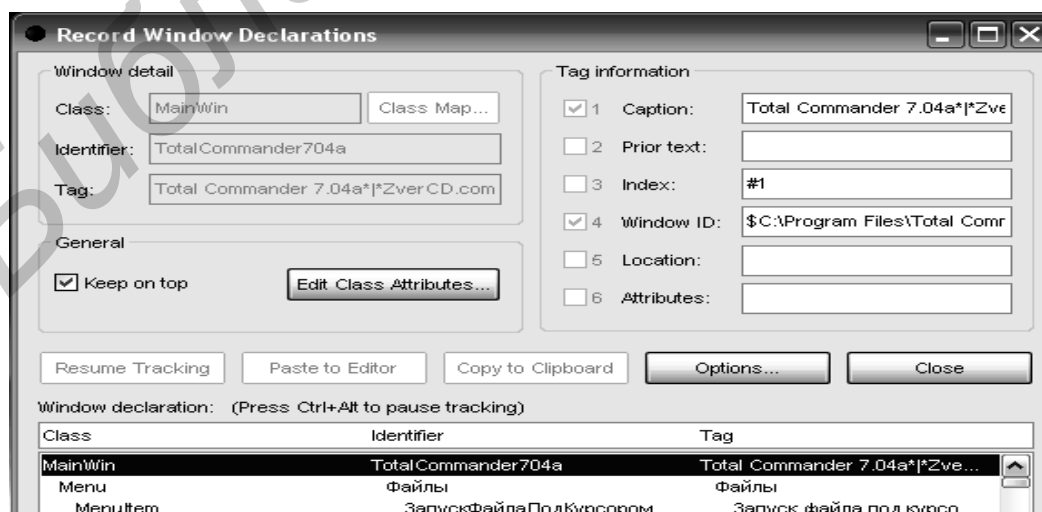


Рис. 3.2

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.

### 3.1.3. 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;
- блок объявления функций. Область для выделения повторяющихся участков программного кода в отдельные именованные блоки;
- тест-кейсы. Модули со сценариями автоматического тестирования. Каждый сценарий помещается в модули с ключевым словом testcase;
- функция main. Точка входа в программу.

## 3.2. 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 распознавания объектов.

### 3.2.1. 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 следующим образом (рис. 3.3):



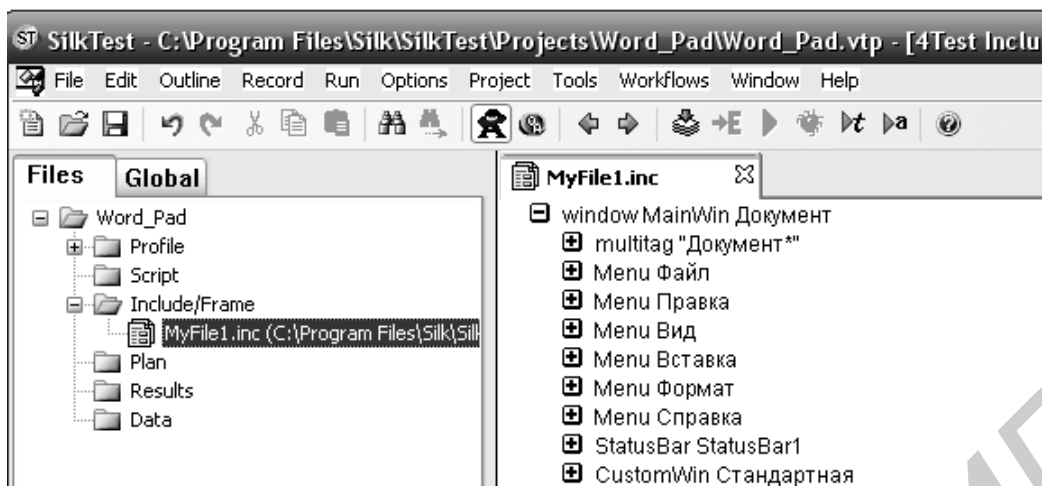


Рис. 3.3

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

### 3.2.2. 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».

Рабочее окно SilkTest будет выглядеть следующим образом (рис. 3.4):

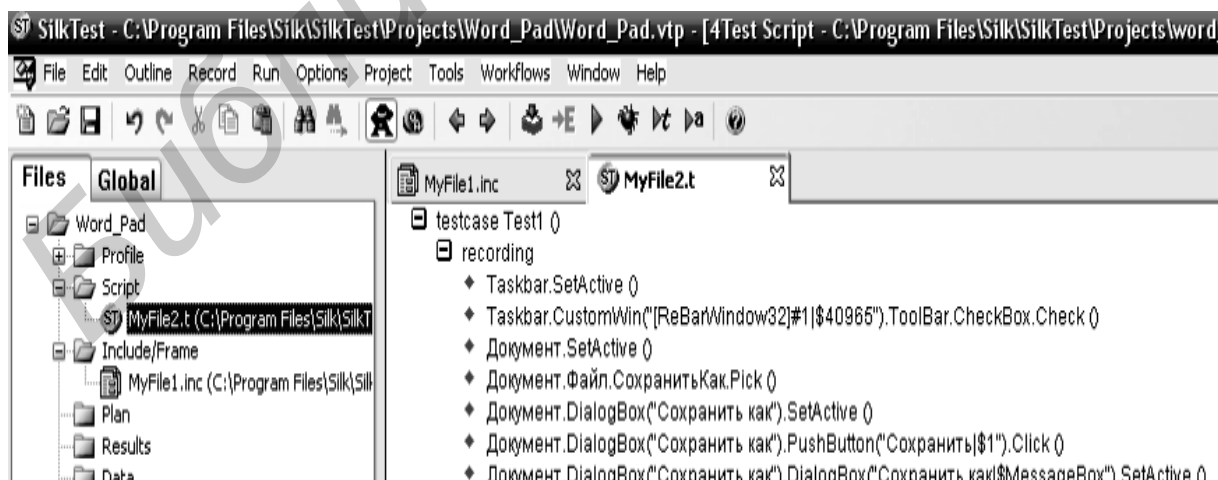



Рис. 3.4

Скрипт, созданный в режиме Recording, можно изменить и дополнить ручным способом. В прил. 1 приведено описание основных функций языка 4Test. В прил. 2 дано описание функций для базовых элементов графического интерфейса.

### 3.2.3. Среда запуска скриптов

На рис. 3.4 представлен фрагмент программного кода, выражающего принятые 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.

### 3.2.4. Средство распознавания объектов

Когда необходимо посмотреть описание конкретного объекта, следует использовать средство распознавания объектов SilkTest. Его можно вызвать следующим образом: Record-Window Identifiers. Чтобы получить информацию об объекте, необходимо навести на него курсор и нажать комбинацию клавиш <Ctrl+Alt>. Описание можно скопировать в файл, нажав кнопку «Paste to Editor» (рис. 3.5):

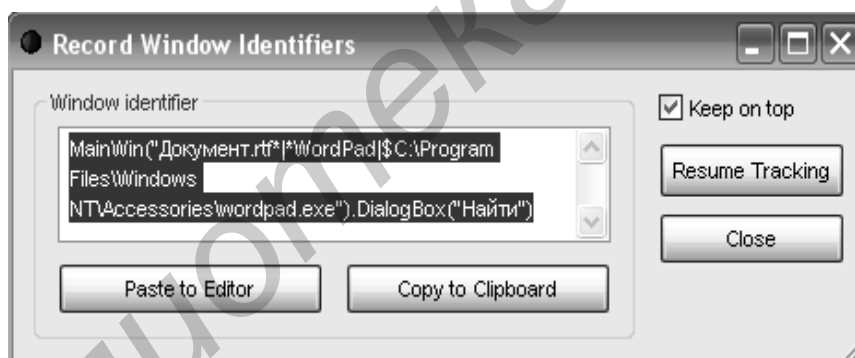


Рис. 3.5

## 3.3. 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 тест-планом), 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 тест-кейса (до `AppState`);
- `TestCaseExit` – выполняется сразу после завершения тест-кейса (после `AppState`);
- `TestPlanEnter` – выполняется сразу перед началом выполнения тест-плана;
- `TestPlanExit` – выполняется сразу после выполнения тест-плана.

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.

### 3.4. Создание тест-плана (Test Plan)

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

Для создания тест-плана следует выбрать пункт меню `File-New-Test Plan`.

#### 3.4.1. Добавление тест-кейса в тест-план

Для того чтобы добавить в тест-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 тест-плане, которые будут отображаться в результатах выполнения тестов.

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

*Script: ScriptFile.t, //файл, содержащий тест-кейсы приложения*

*Testcase: CurrentAppStateTest //тест, включаемый в тест-план*

*Testdata:"01000101" //параметр для тест-кейса CurrentAppStateTest*

*Comment: "Проверка текущих параметров приложения"*

### 3.4.2. Запуск тест-плана (полный, выборочный)

Для запуска тест-плана на выполнение необходимо выбрать пункт меню Run-Run All Testcases или нажать клавишу <F9>. Существует возможность выборочного запуска тест-кейсов из тест-плана. 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.

## 3.5. 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.

Перечислим случаи, при которых целесообразно использовать 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;

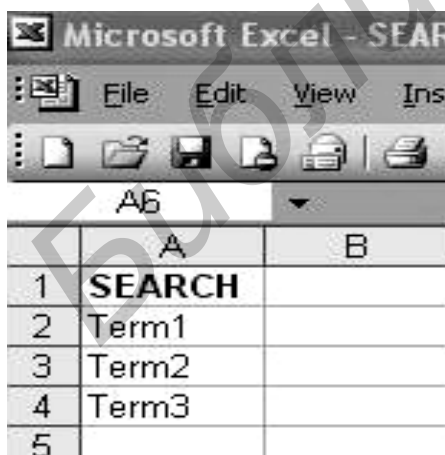
- для непосредственного доступа к базе данных тестируемого приложения. Например, для проверки того, что по определенному запросу в приложении выводится необходимый набор данных. Для этого можно считать данные из базы данных, а затем сравнить их с теми, что выводятся при работе с приложением;

- 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:



The image shows a screenshot of a Microsoft Excel spreadsheet. The title bar reads 'Microsoft Excel - SEAR'. The menu bar includes 'File', 'Edit', 'View', and 'Ins'. The toolbar contains icons for file operations. The active cell is A6. The spreadsheet content is as follows:

	A	B
1	<b>SEARCH</b>	
2	Term1	
3	Term2	
4	Term3	
5		

Рис. 3.6

Шаг 1: создать Excel-файл с именем Search и заполнить Excel-таблицу данными для поиска (рис. 3.6).

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

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

Шаг 4: открыть окно Select Testcase. Выбрать тест-кейс и нажать «Ok» (рис. 3.7).

Шаг 5: открыть окно Specify Data Driven Script. Выбрать «Create a new file/Overwrite an existing file» и нажать «Ok» (рис. 3.8).

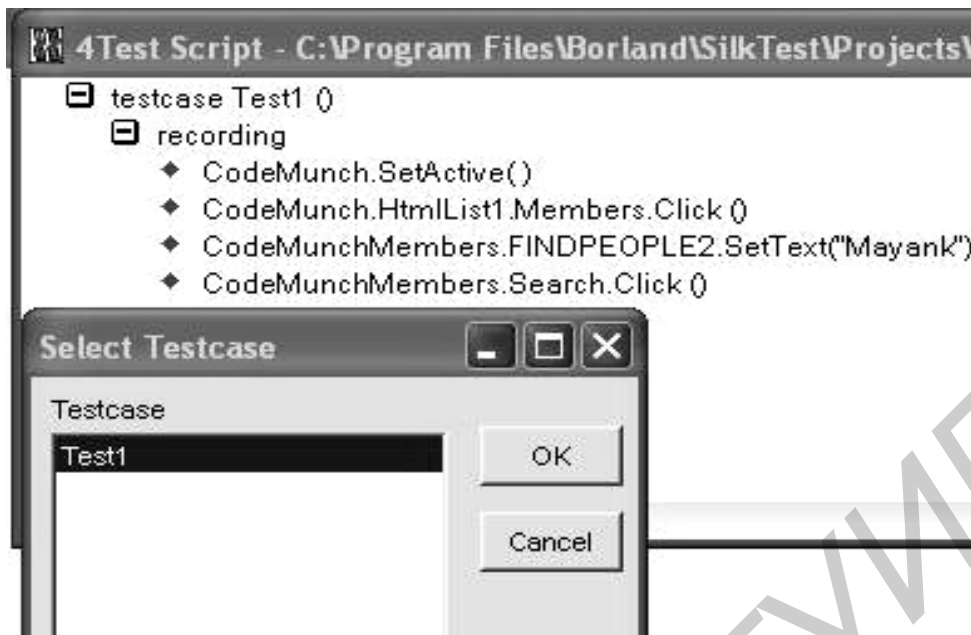


Рис. 3.7

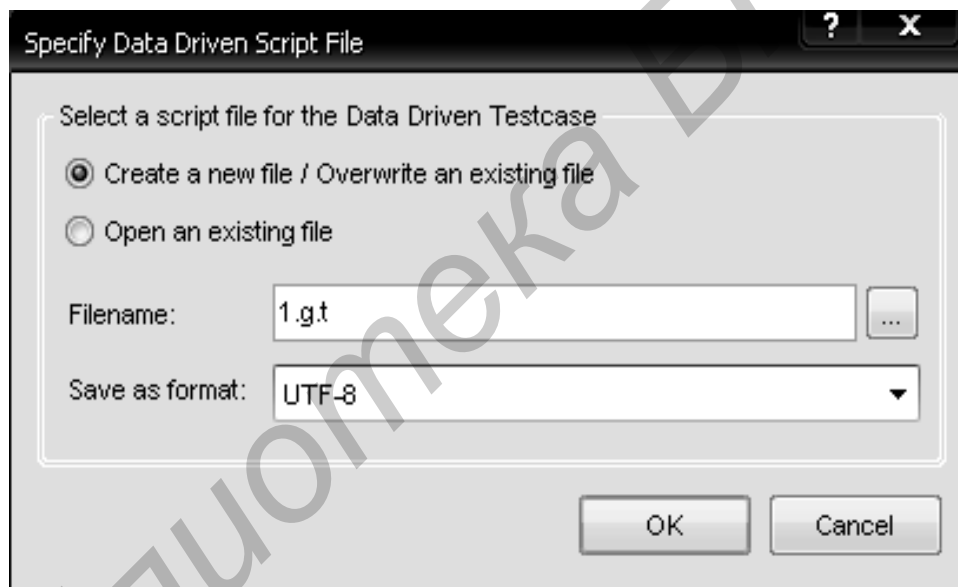


Рис. 3.8

Шаг 6: открыть окно Select Data Source. Выбрать Silk DDA Excel. Выбрать Excel-файл через диалог Browse и нажать «Ok» (рис. 3.9).

Шаг 7: открыть Specify Data Driven Testcase окно. Выбрать Add a new Data Driven Testcase и нажать «Ok» (рис. 3.10).

Шаг 8: открыть окно Find/Replace Values. Нажать «Cancel» в Find/Replace Values (рис. 3.11).

Шаг 9: войти в файл, сохраненный с именем скрипта, но с расширением \*.g.t (рис. 3.12).

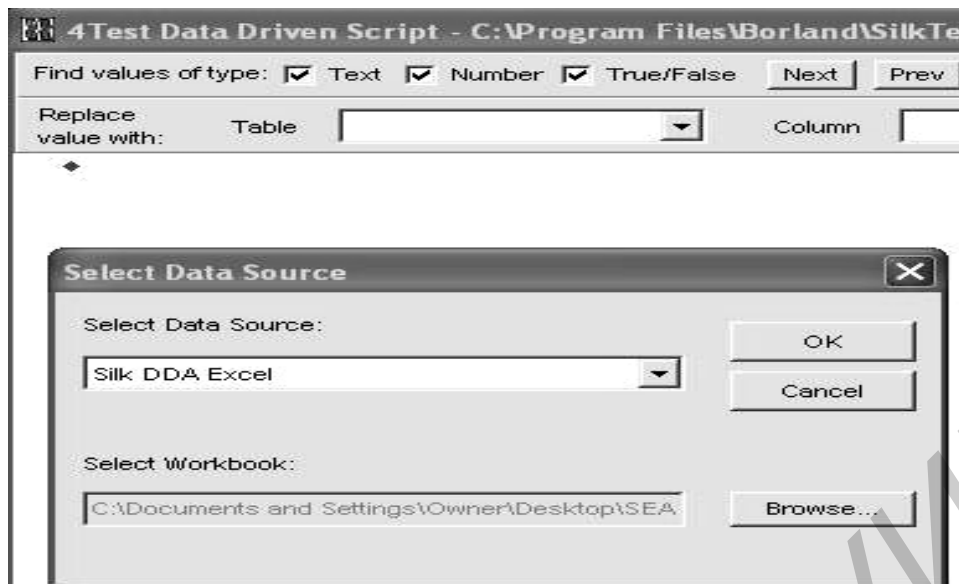


Рис. 3.9

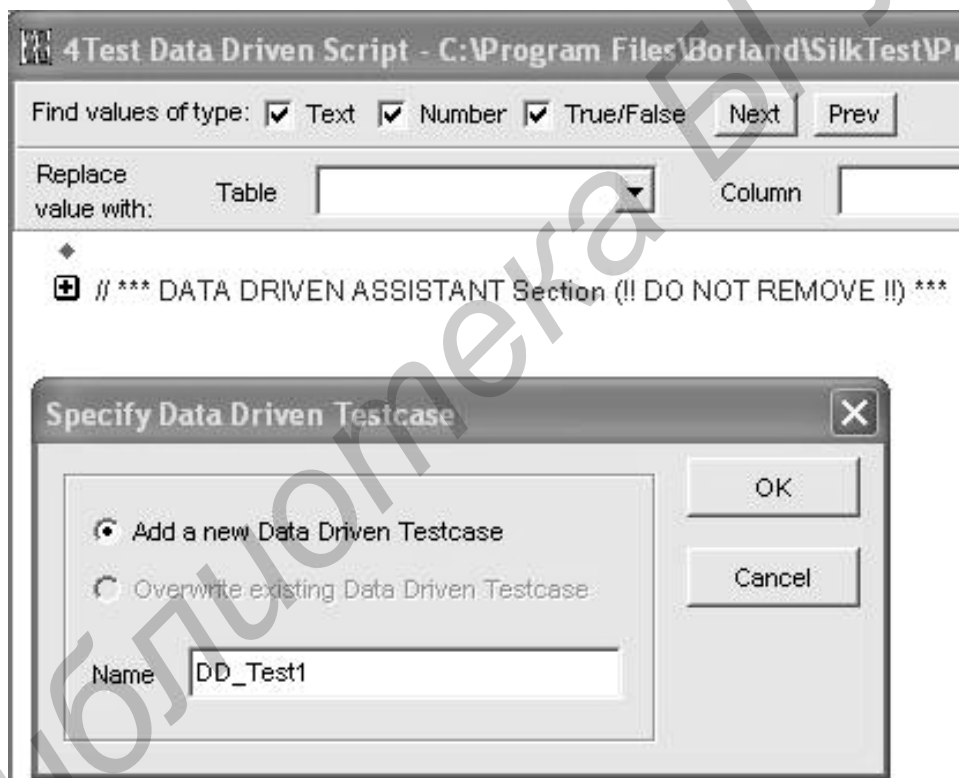


Рис. 3.10

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 (рис. 3.13).

Шаг 12: нажать кнопку «Replace» (рис. 3.14).

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

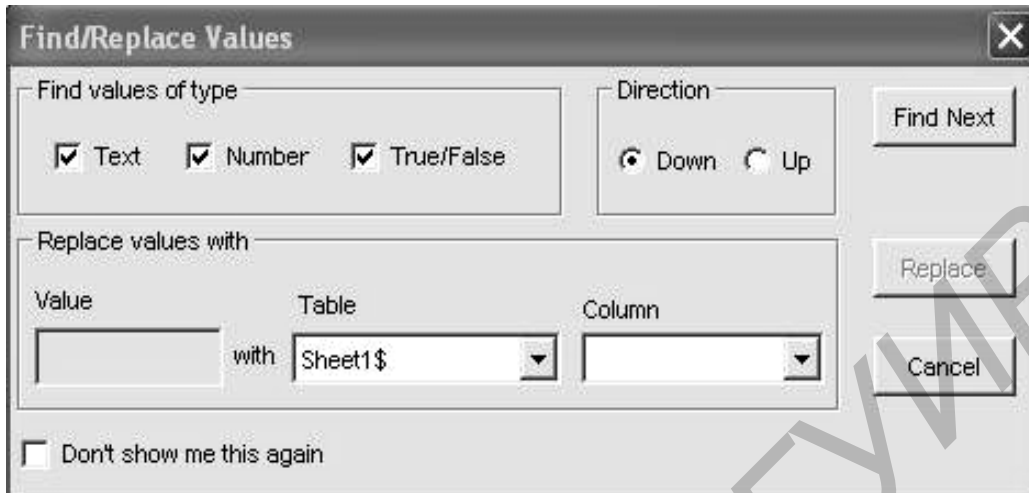


Рис. 3.11



Рис. 3.12

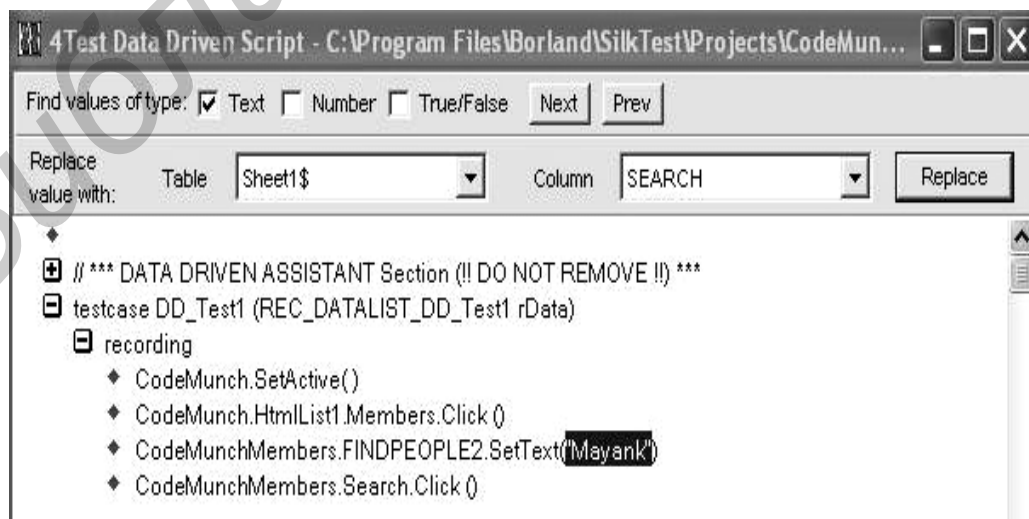


Рис. 3.13



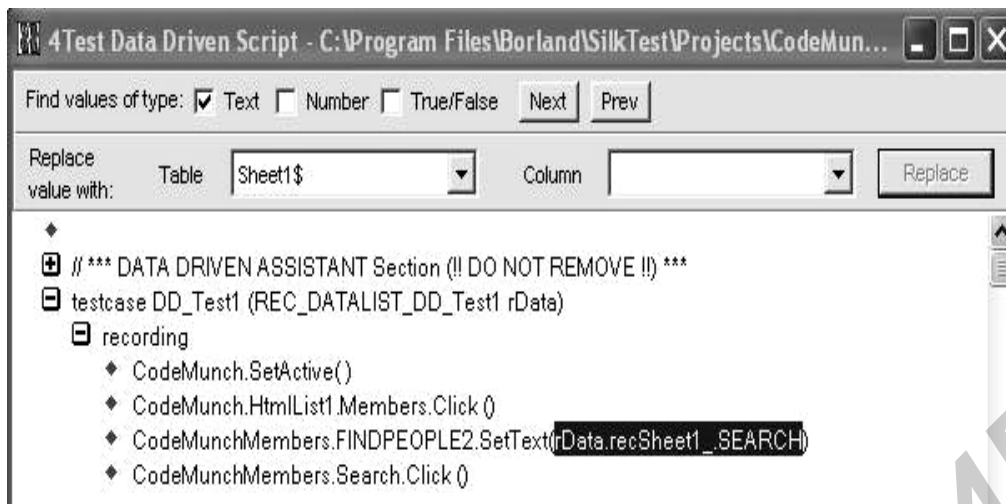


Рис. 3.14

### 3.6. Просмотр результатов исполнения тестов

После запуска тест-кейса, тест-плана 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 (рис. 3.15).

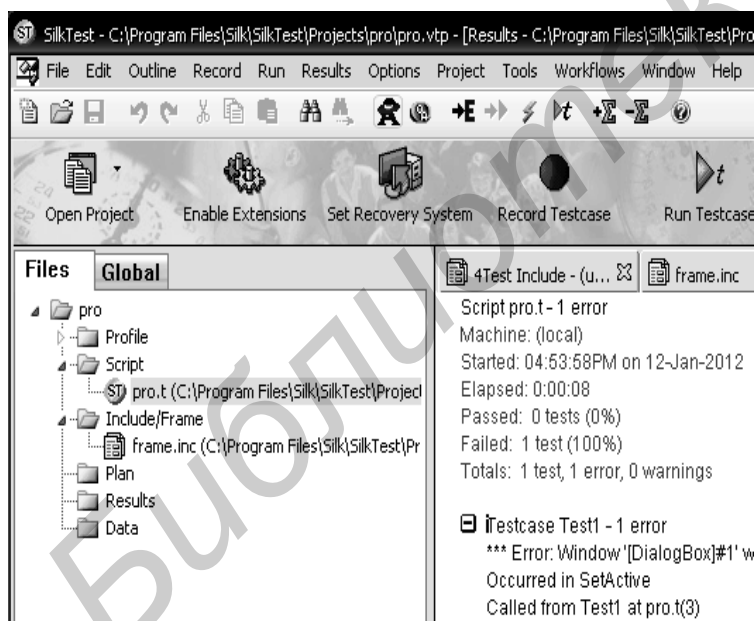


Рис. 3.15

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

- History size – количество хранящихся результатов, выполненных тест-кейсов. По умолчанию хранятся только 5 последних результатов (значение по умолчанию может быть изменено до 32 767);

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

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.

– 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 записываться 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 тест-плана. Данная опция доступна лишь в том случае, если файл результатов был сгенерирован при запуске отдельного тест-кейса или скрипта (но не другого тест-плана);

- 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 тест-плане только те тест-кейсы, которые прошли с ошибками, после чего можно их перезапустить отдельно от остальных.

### 3.7. Пример разработки теста

#### 3.7.1. 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».

##### 3.7.1.1. 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.**

##### 3.7.1.2. 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 (1111 1111 1111 0110).
15. Набрать второй операнд (1011).
16. Нажать клавишу +/- для смены знака.
17. Проверить результат (-1011).
18. Нажать клавишу «=».
19. Сравнить результаты. **Ожидаемый результат: 1111 1111 1110 1011.**

#### 3.7.1.3. 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).

#### 3.7.1.4. Дополнительные требования к созданию тестов

1. Разработать следующие тестовые планы (Test Plan):
  - 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:

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.

### 3.7.2. Создание фрейма приложения

На первом шаге необходимо создать новый проект, куда будут помещаться далее разработанные скрипты и тест-планы. Для этого в SilkTest следует выбрать пункт меню File-New Project и указать Classic Agent. Нажать «Ok» (рис. 3.16).

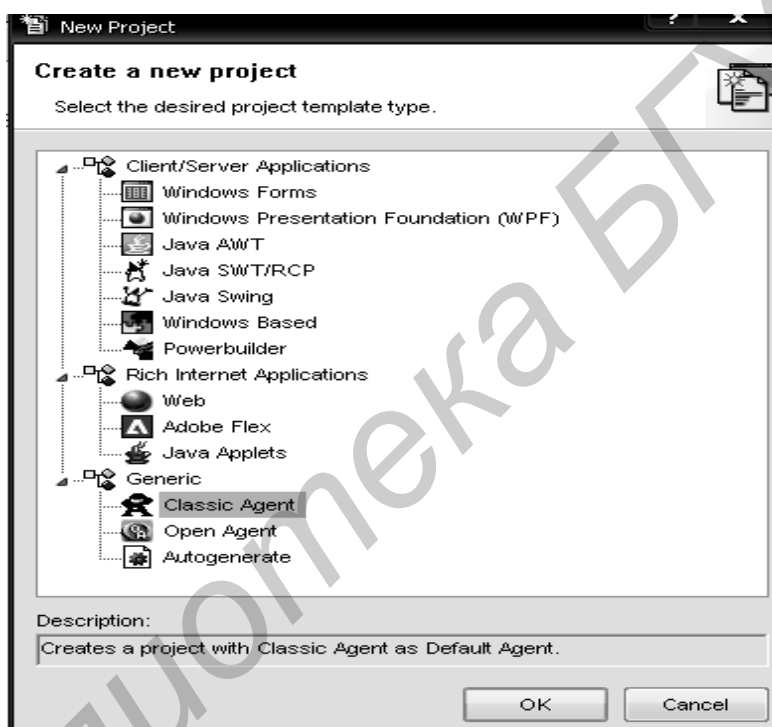


Рис. 3.16

После задания имени проекта, например Add, откроется окно (рис. 3.17), в которое можно помещать описания окон и скрипты.

Теперь необходимо подготовить тестовое 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 рис. 3.18.



Рис. 3.17

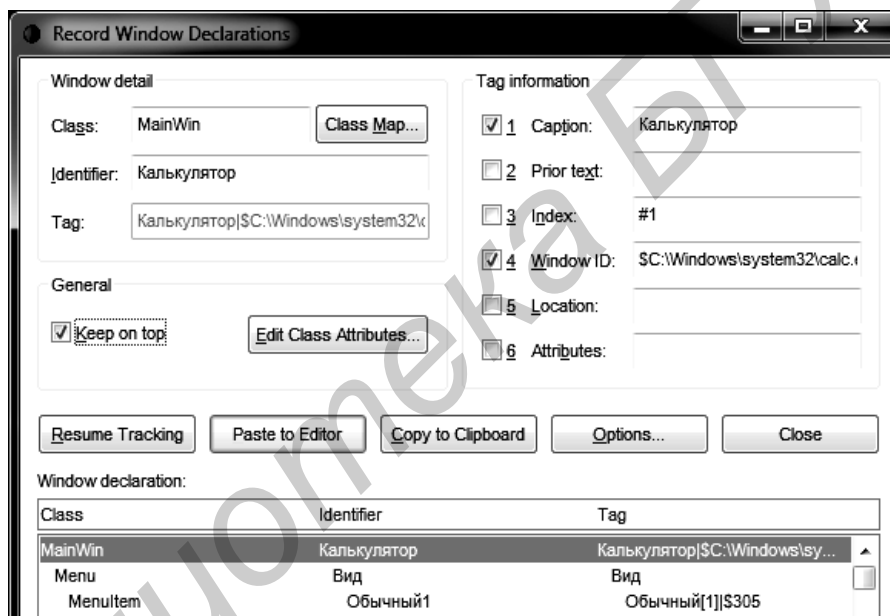


Рис. 3.18

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 которой 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 (рис. 3.19). 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 привести 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 (рис. 3.20) 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».

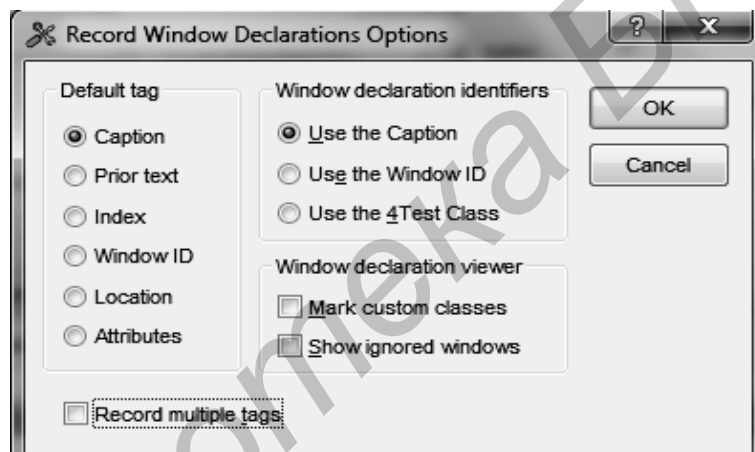


Рис. 3.19

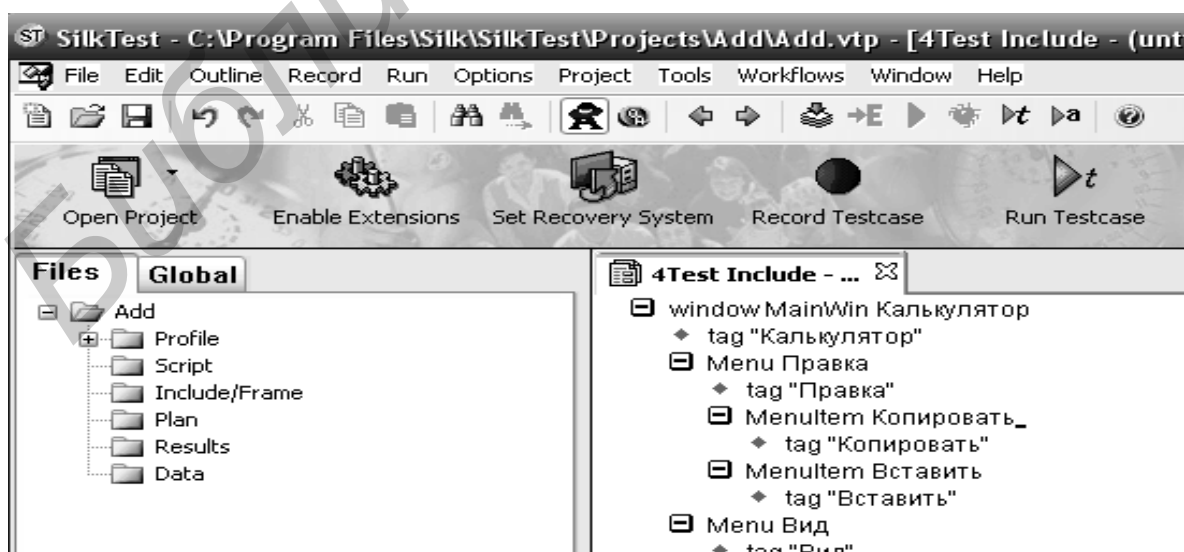


Рис. 3.20

### 3.7.3. Настройка Recovery-системы

Для настройки Recovery-системы необходимо выбрать пункт «Set Recovery System». В появившемся окне выбрать тестируемое приложение «Калькулятор» и нажать кнопку «Ок» (рис. 3.21). 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 (рис. 3.22).

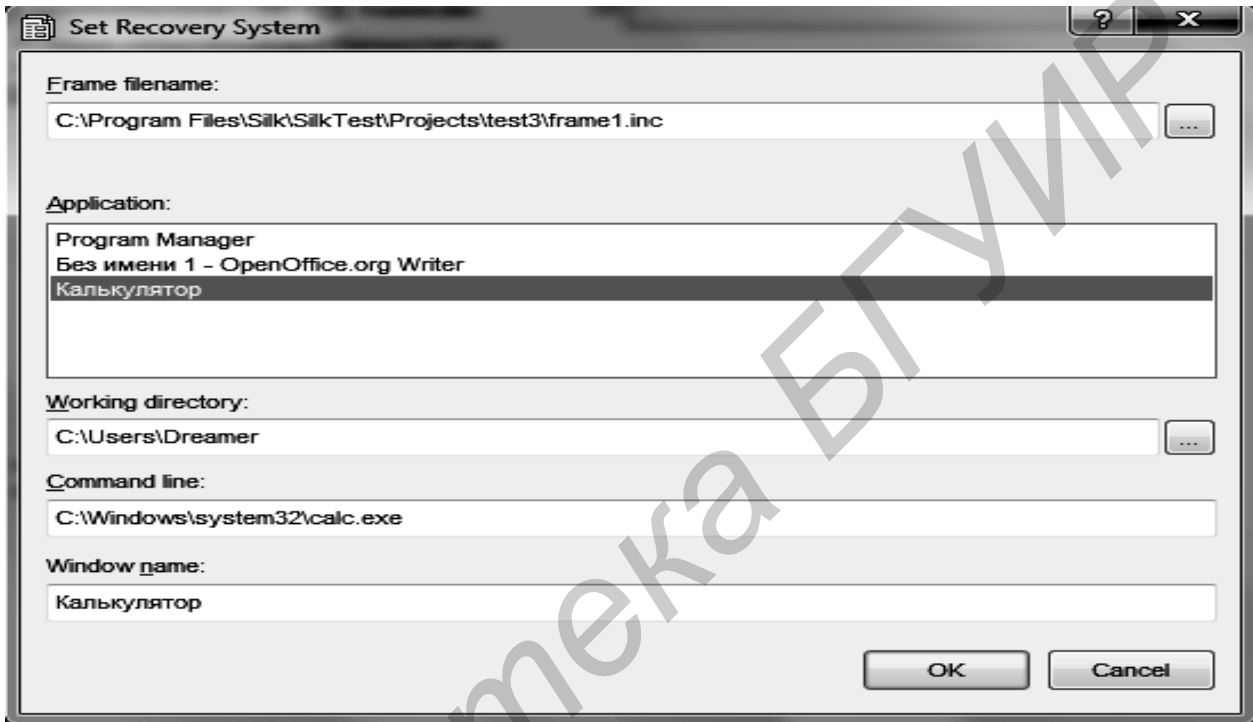


Рис. 3.21

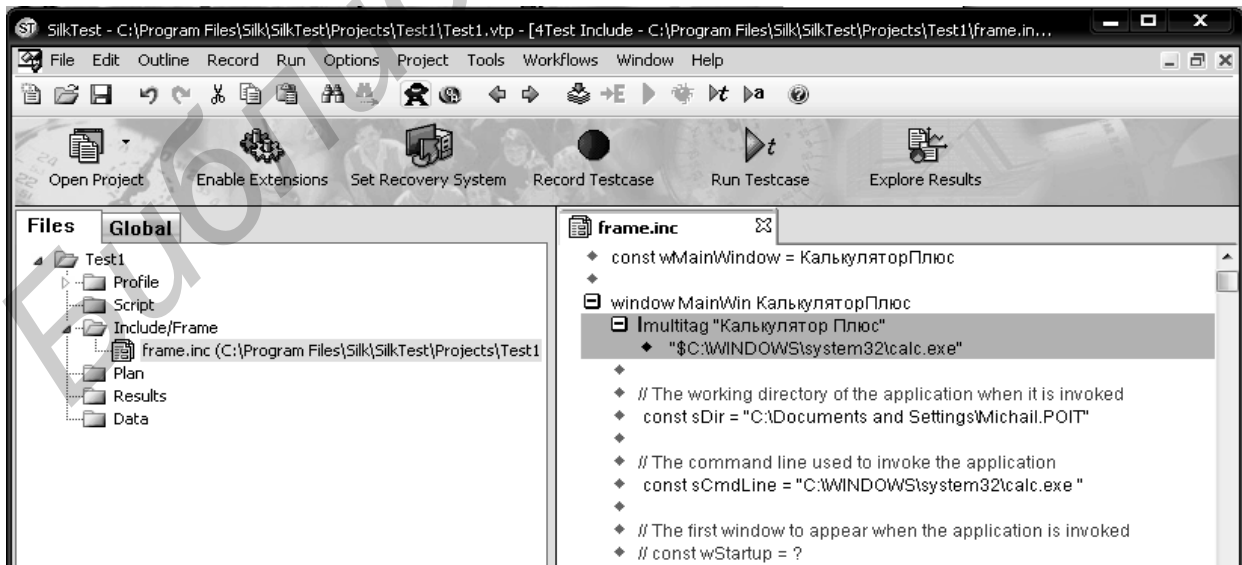


Рис. 3.22



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

### 3.7.4. Запись и воспроизведение скрипта

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

#### 3.7.4.1. Создание скрипта для тест-кейса №1

Для создания тест-кейса следует выбрать пункт меню Record-Testcase (рис. 3.22). В поле Testcase name открывшегося окна ввести имя теста «Test1». Нажать кнопку «Start Recording» (рис. 3.23).

На экране в правом нижнем углу появится окошко Record Status, отображающее статус записи и текущее окно под курсором мыши (рис. 3.24).

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



Рис. 3.23

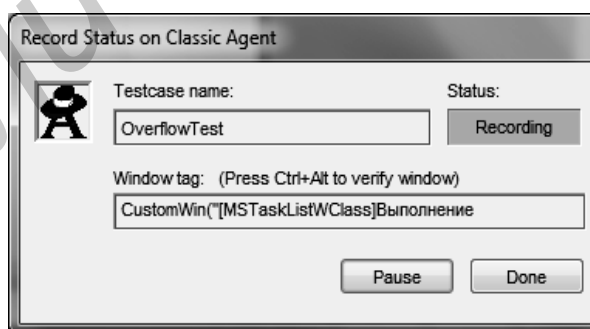


Рис. 3.24

Находясь в режиме записи, выполним следующие действия:

1. Переключение приложения на вид «Программист».
2. Переключение на операцию «Vin-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» (рис. 3.25).

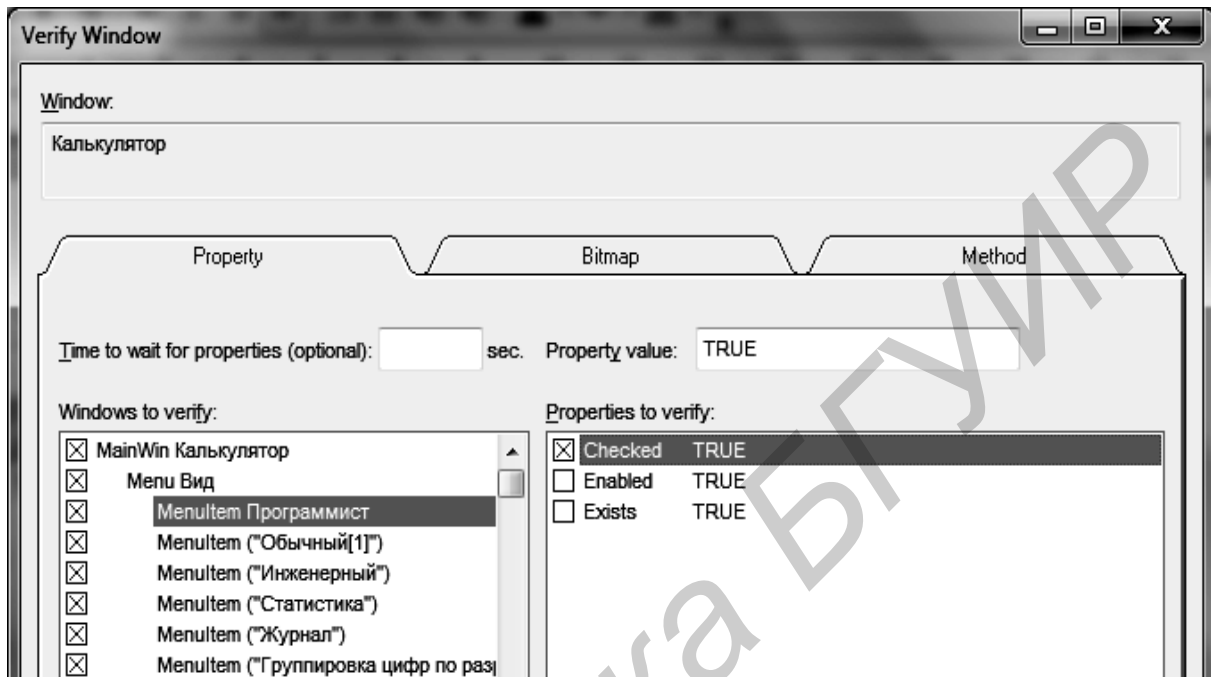


Рис. 3.25

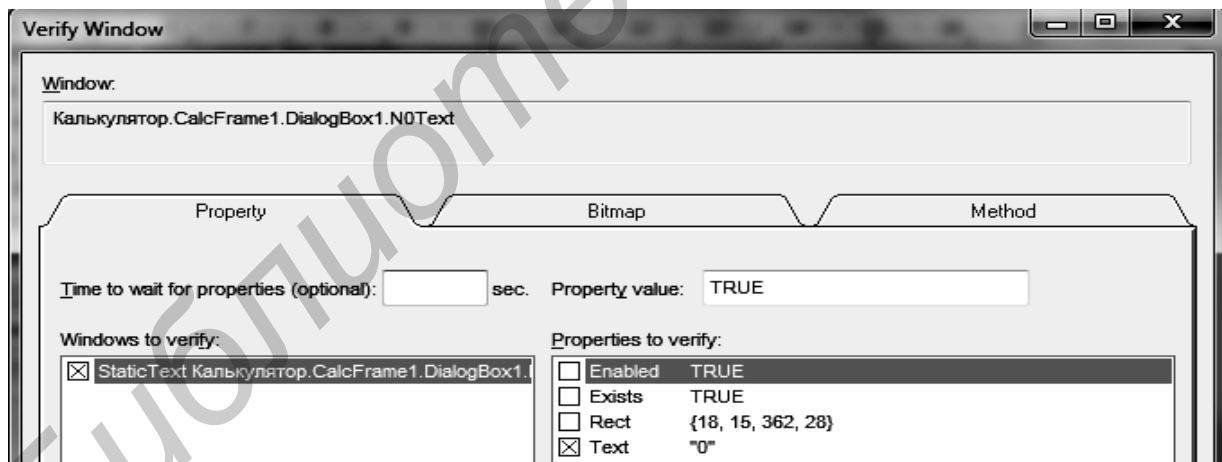


Рис. 3.26

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 рис. 3.26.

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 ()
  [ ] 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 информационным 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 1с и SilkTest выдает отчет о проделанной работе: время начала работы скрипта, как он долго работал, сколько произошло ошибок и предупреждений (рис. 3.27).

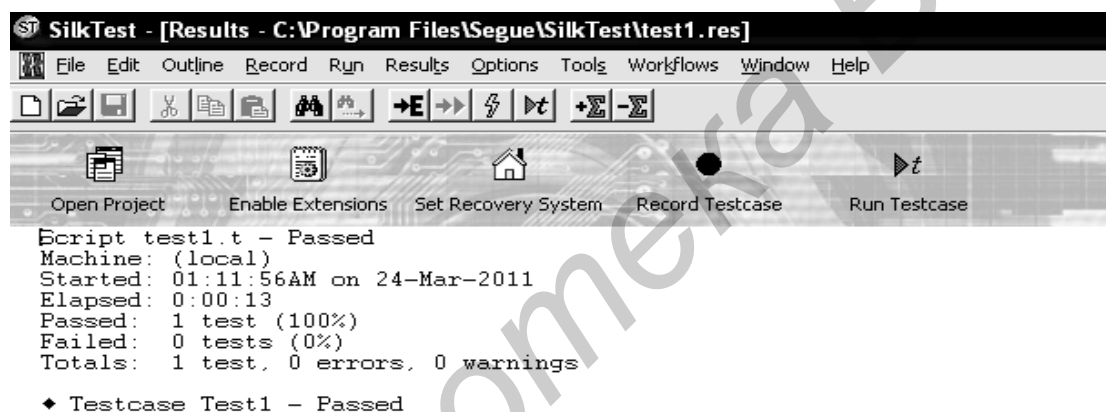


Рис. 3.27

#### 3.7.4.2. 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"}

```

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

Для создания тест-кейса №3 следует использовать технологию Data Driven Testing. Но сначала необходимо создать тест-кейс без использования Data Driven Testing, 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

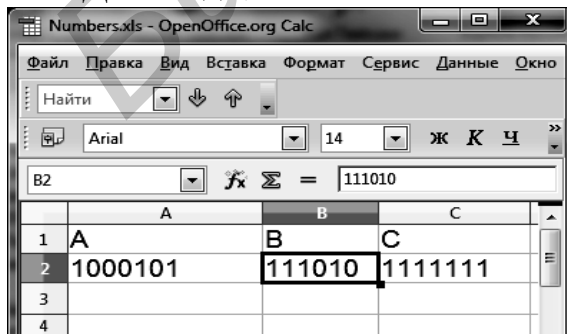
```

```

[ ] 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 следующие шаги:



Шаг 1: создать файл с именем Numbers и заполнить Excel-таблицу данными для тест-кейса (рис. 3.28).

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

Шаг 3: выбрать Data Driven Testcase.

Рис. 3.28

Шаг 4: открыть окно Select Testcase. Выбрать тест-кейс Test3 (рис. 3.7).

Шаг 5: открыть окно Specify Data Driven Script File. Выбрать «Create a new file/Overwrite an existing file» и нажать «Ok» (рис. 3.8).

Шаг 6: открыть окно Select Data Source. Выбрать Silk DDA Excel. Выбрать Excel-файл через диалог Browse и нажать «Ok» после выбора (рис. 3.9).

Шаг 7: открыть Specify Data Driven TestCase окно. Выбрать Add a new Data Driven Testcase и нажать «Ok» (рис. 3.10).

Шаг 8: открыть окно Find/Replace Values. Нажать Cancel в Find/Replace Values (рис. 3.11).

Шаг 9: войти в файл с именем скрипта, но с расширением \*.g.t (рис. 3.12).

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 (рис. 3.13).

Шаг 11: выбранные значения вставить в скрипт (рис. 3.14).

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_DATALIST_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)}

```

### 3.7.5. Создание тест-плана

Так как в проекте создано несколько тест-кейсов, их выполнение можно организовать в тест-планах 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 тест-плана следует выбрать пункт меню File-New-Test Plan (рис. 3.1). В результате будет создан пустой тест-план в виде файла с расширением \*.pln. Для добавления тест-кейсов в тест-план следует выбрать пункт TestPlan Detail (рис. 3.29).

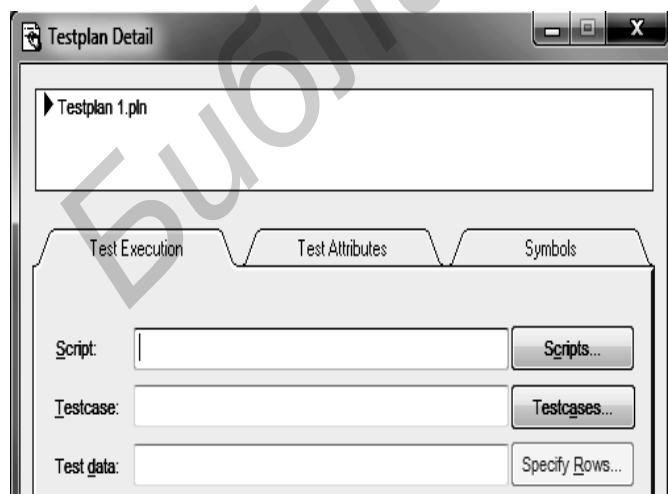


Рис. 3.29

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 «Ок». В результате будет получен тест-план запуска тестов (рис. 3.30). Для запуска тест-плана необходимо выбрать пункт Run-All Tests либо кнопку на панели управления SilkTest-Run the Script,

suite or Testplan the Active Window.

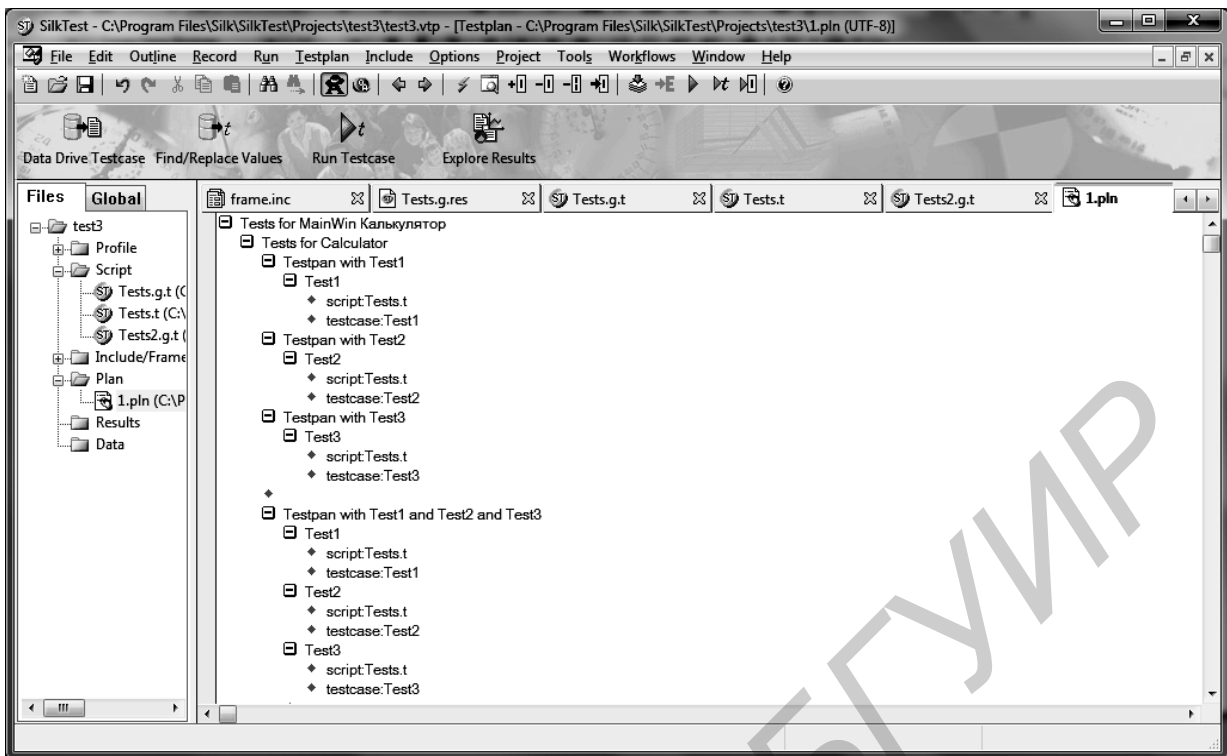


Рис. 3.30

### 3.8. Редактирование текста скрипта

Редактор SilkTest отличается от большинства других редакторов кода. Дело в том, что в отличие от других языков программирования в языке 4Test используется иерархическое представление блоков кода.

Для выделения блока кода (например условный блок `if` или циклы `for`), необходимо поместить его на один уровень «правее» в редакторе. Точно так же можно скрывать длинные комментарии, оставляя видимой только первую строку и раскрывая весь комментарий по мере необходимости.

«Передвигать» блоки кода можно с помощью комбинаций клавиш `<Alt+стрелочки>` (вверх, вниз, влево или вправо). Чтобы «раскрыть» блок, следует нажать `<Ctrl+цифровой плюс>`. `<Ctrl+цифровой минус>`, наоборот, свернет блок кода.

Если поместить блок кода в недопустимом месте, то SilkTest пометит эти строки красными крестиками, а в строке статуса выдаст сообщение `Syntax error`.

Для примера рассмотрим удаление ключевого слова `Recording` в тексте программного кода:

- поставить курсор на строку, содержащую слово `Recording`, и нажать комбинацию клавиш `<Shift+стрелка вниз>`;
- нажать клавишу `<Delete>`;
- выделить все строки, которые были внутри блока `Recording`, нажать комбинацию клавиш `<Alt+стрелка влево>`.

В результате выделенный блок переместится на один уровень влево.

### 3.9. Задания для выполнения лабораторной работы

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 тест-плана запуска тест-кейсов. Общие для тест-кейсов шаги инструкций поместить в Recovery-файл.

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

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

1. Выбрать пункт меню «Вид–Инженерный».
2. Проверить доступность кнопок «Ave», «Sta», «7», «8».
3. Проверить, отмечены ли чекбоксы «Inv» и «Нур».
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-й 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-й 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-й системе счисления.
7. Вернуть вид приложения «Вид–Обычный».
8. Закрыть окно «Калькулятор».

**Задание №3.** Тест-приложение «Калькулятор». Вид «классический».

Тестирование функции: инженерные расчеты во 2-й 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 во 2-й системе счисления.
7. Вернуть вид приложения «Вид–Обычный».
8. Закрыть окно «Калькулятор».

**Задание №4.** Тест-приложение «Калькулятор». Вид «программист».

Тестирование функции: инженерные расчеты в 16-й 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-й системе счисления.
7. Вернуть вид приложения «Вид–Обычный».
8. Закрыть главное окно «Калькулятор».

**Задание №5.** Тест-приложение «Калькулятор». Вид «программист».

Тестирование функции: инженерные расчеты во 2-й 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 во 2-й системе счисления.
7. Вернуть вид приложения «Вид–Обычный».
8. Закрыть окно «Калькулятор».

**Задание №6.** Тест-приложение «Калькулятор». Вид «программист».

Тестирование функции: статистические расчеты во 2-й 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 во 2-й системе счисления.
7. Вернуть вид приложения «Вид–Обычный».
8. Закрыть окно «Калькулятор».

**Задание №7.** Тест-приложение «Калькулятор». Вид «классический».

Тестирование функции: статистические расчеты в 10-й 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-й системы счисления.

7. Протестировать функции «RET», «LOAD».

8. Вернуть вид приложения «Вид–Обычный».

9. Закрыть окно «Калькулятор».

**Задание №8.** Тест-приложение «Калькулятор». Вид «классический».

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

7. Протестировать функции «CD», «CAD».

8. Вернуть вид приложения «Вид–Обычный».

9. Закрыть окно «Калькулятор».

**Задание №9.** Тест-приложение «Калькулятор». Вид «классический».

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

7. Вернуть вид приложения «Вид–Обычный».

8. Закрыть окно «Калькулятор».

**Задание №10.** Тест-приложение «Калькулятор». Вид «классический».

Тестирование функции: работа с памятью во 2-й 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 во 2-й системе счисления.

7. Вернуть вид приложения «Вид–Обычный».

8. Закрыть окно «Калькулятор».

## ЛИТЕРАТУРА

1. Котляров, В. П. Основы тестирования программного обеспечения / В. П. Котляров, Т. В. Коликова. – М. : Интернет-университет информационных технологий, 2006. – 285 с.
2. Дастин, Э. Автоматизированное тестирование программного обеспечения Automated Software Testing / Э. Дастин, Дж. Рэшка. – М. : Лори, 2005. – 592 с.
3. Браун, К. Быстрое тестирование / К. Браун, Р. Калбертсон, Г. Кобб. – М. : Вильямс, 2002. – 384 с.
4. Бейзер, Б. Тестирование черного ящика. Технологии функционального тестирования ПО. Black-Box Testing. Techniques for Functional Testing of Software and Systems. Сер. Библиотека программиста / Б. Бейзер. – СПб : Питер, 2004. – 320 с.
5. Роббинс, Дж. Отладка приложений. Сер. Мастер / Дж. Роббинс. – М. : BHV, 2001. – 512 с.
6. Тамре, Л. Введение в тестирование программного обеспечения / Л. Тамре. – М. : Вильямс, 2003. – 359 с.
7. Макгрегор, Дж. Тестирование объектно-ориентированного программного обеспечения / Дж. Макгрегор, Д. Сайкс. Киев : ТИД «Диасофт», 2002. – 432 с.
8. Савин, Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах / Р. Савин. – М. : Дело, 2007. – 312 с.
9. Винниченко, И. Автоматизация процессов тестирования. Сер. Библиотека программиста / И. Винниченко. – СПб : Питер, 2005. – 202 с.
10. Куликов, С. С. Тестирование ПО, 2-е изд., переработанное и дополненное / С. С. Куликов. – Минск : ЗАО «БелХард Групп» Центр обучающих технологий, 2008. – 202 с.
11. Канер, С. Фундаментальные концепции менеджмента бизнес-приложений / С. Канер, Дж. Фолк, Е. К. Нгуен. – Киев: ТИД «Диасофт», 2001. – 544 с.
12. QuickTest Professional User's Guide Version 6.5: Mercury Interactive Corporation [Электронный ресурс]. – Режим доступа: <http://www.learnqtp.com/qtp-quality-center-test-director-guide/>. Дата доступа 06.04.2011.
13. Software Test Engineering with IBM Rational Functional Tester: The Definitive Resource / Chip Davis, Lee B. Thomas. – IBM Press, 2009. – 69p.
14. Automated Software Regression Testing & Functional Software Testing – from Borland [Электронный ресурс]. – 2011.– Режим доступа: <http://www.borland.com/us/products/silk/silktest/>. Дата доступа 06.04.2011.
15. Automated Testing - TestComplete Features [Электронный ресурс]. – 2011. – Режим доступа: <http://smartbear.com/products/qa-tools/automated-testing/>. Дата доступа 03.01.2012
16. Selenium Tutorial for Beginner/Tips for Experts [Электронный ресурс]. – 2011. – Режим доступа <http://seleniumhq.org/>. Дата доступа 09.04.2011.

ОПИСАНИЕ ЯЗЫКА 4TEST

П.1.1. Типы данных

П.1.1.1. 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 (табл. П.1.1); *variable-id* – имя переменной; *expr* – инициализирующее выражение.

Таблица П.1.1

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	

П.1.1.2. 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.

### П.1.1.3. Примеры определения и инициализации переменных

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

```
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"
```

### П.1.1.4. 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).

## П.1.2. Операторы и функции языка 4Test

### П.1.2.1. Арифметические операторы

Арифметические операторы, поддерживаемые 4Test, представлены в табл. П.1.2.

Таблица П.1.2

Арифметические операторы

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



### П.1.2.2. Логические операторы

Логические операторы, поддерживаемые языком 4Test, представлены в табл. П.1.3.

Таблица П.1.3

#### Логические операторы

Оператор	Значение
&&	И
	ИЛИ
!	Не

### П.1.2.3. Операторы сравнения

Операции отношения, поддерживаемые 4Test, представлены в табл. П.1.4.

Таблица П.1.4

#### Операции отношения

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

### П.1.2.4. 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]
```

### П.1.2.5. 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**

#### П.1.2.6. Функции приведения типа

Функции приведения типа языка 4Test представлены в табл. П.1.5.

Таблица П.1.5

Функции приведения типа

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

#### П.1.2.7. Функции работы с массивами

Функции работы с массивами языка 4Test представлены в табл. П.1.6.

Таблица П.1.6

## Функции работы с массивами

Наименование функции	Выполняемое действие	Параметры функции
<i>ArrayFind</i> ( <i>aArray</i> , <i>aElem</i> [, <i>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</i> ( <i>aArray</i> , <i>iNewSize</i> [, <i>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</i> ( <i>aArray</i> [, <i>iDim</i> ])	Функция возвращает количество элементов массива	<i>aArray</i> – имя массива; <i>iDim</i> – измерение массива
<i>ArraySort</i> ( <i>aArray</i> [, <i>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 табл. П.1.7.

Таблица П.1.7

## Функции работы со списками

Наименование функции	Выполняемое действие	Параметры функции
<i>ListAppend</i> ( <i>IList</i> , <i>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</i> ( <i>IList</i> )	Функция возвращает количество элементов в списке.	<i>IList</i> – имя списка
<i>ListDelete</i> ( <i>IList</i> , <i>iIndex</i> )	Функция удаляет указанный элемент списка	<i>IList</i> – имя списка; <i>iIndex</i> – искомый элемент
<i>ListInsert</i> ( <i>IList</i> , <i>index</i> , <i>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</i> ( <i>lOrig</i> , <i>lMerge</i> [, <i>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</i> ( <i>IList</i> , <i>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</i> ( <i>IList</i> , <i>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</i> ( <i>IList</i> )	Функция сортирует список <i>IList</i>	<i>IList</i> – имя списка
<i>ListPrint</i> ( <i>IList</i> )	Функция распечатывает список <i>IList</i> в файл результатов	<i>IList</i> – имя списка

### П.1.2.8. Функции работы со строками

Функции работы со строками языка 4Test представлены в табл. П.1.8.

Таблица П.1.8

#### Функции работы со строками

Наименование функции	Выполняемое действие	Параметры функции
1	2	3
<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 самого левого символьного значения	<i>sString</i> – имя строки; <i>iNumChars</i> – количество возвращаемых 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

## Окончание табл. П.1.8

1	2	3
<i>MatchStr</i> ( <i>sPattern</i> , <i>sString</i> )	Функция проверяет строку на присутствие «маски» поиска	<i>sPattern</i> – «маска» поиска; <i>sString</i> – искомая строка
<i>SubStr</i> ( <i>sString</i> , <i>iPos</i> [, <i>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</i> ( <i>sSubstr</i> , <i>sTarget</i> [, <i>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 ( <i>TRUE</i> , если поиск производится с конца, не указывается, если с начала)
<i>StrTran</i> ( <i>sOrig</i> , <i>sSearch</i> , <i>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</i> ( <i>sOrig</i> , <i>iPos</i> , <i>iLen</i> , <i>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</i> ( <i>sString</i> )	Функция возвращает длину строки	<i>sString</i> – имя строки

## ЭЛЕМЕНТЫ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ И ИХ ФУНКЦИИ

## П.2.1. 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).

## П.2.2. Функции элементов графического интерфейса

## П.2.2.1. Функции для Window (окно)

Функции элемента Window:

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

## П.2.2.1.1. Активирование окна

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

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

Пример. Следующая строка кода активирует окно Window\_1:

*Window\_1.SetActive()*

## П.2.2.1.2. 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()*

## П.2.2.1.3. Разворачивание окна

Иногда бывает необходимо развернуть ранее свернутое окно:

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

Пример. Разворачивание окна Window\_1:

*Window\_1.Maximize()*

#### П.2.2.2. 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:

*Menu.Option1.Pick()*

#### П.2.2.3. 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.

##### П.2.2.3.1. 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")*

##### П.2.2.3.2. 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 CreditCard = (Visa, Master Card, American Express):

*CreditCard.Select("#0")*

##### П.2.2.3.3. 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.

П.2.2.3.4. 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.

П.2.2.3.5. 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")*

П.2.2.3.6. 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")*

П.2.2.3.7. 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()*

П.2.2.3.8. 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()
```

#### П.2.2.3.9. 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()
```

#### П.2.2.4. Функции для *Button* (Кнопка)

Для элемента *Button* предусмотрены следующие функции:

- нажатие кнопки;
- получение сведений о состоянии элемента.

##### П.2.2.4.1. Нажатие кнопки

Нажатие кнопки реализует следующая функция

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

Пример. Нажатие кнопки *Submit*:

```
Submit.Click()
```

##### П.2.2.4.2. 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)

#### П.2.2.5. Функции для *CheckBox/RadioButton* (Флажок/Переключатель)

Для элементов *CheckBox/RadioButton* предусмотрены следующие функции:

- выбор параметра;
- отмена выбранного параметра;

- получение информации о состоянии элемента.

#### П.2.2.5.1. 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")
```

#### П.2.2.5.2. 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")
```

#### П.2.2.5.3. 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")
```

#### П.2.2.6. Функции для EditBox (Поле ввода)

Для элемента EditBox предусмотрены следующие функции:

- ввод информации в поле ввода;
- удаление информации;

- получение значения из поля ввода.

#### П.2.2.6.1. 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")
```

#### П.2.2.6.2. 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()
```

#### П.2.2.6.3. 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()
```

#### П.2.2.7. 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.

### П.2.3. 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 нажа-

том 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>")
```

#### П.2.4. 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)
```

#### П.2.5. 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")
```

## П.2.6. 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))
```

## П.2.7. 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.

## П.2.8. 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()
```

## П.2.9. 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, "1010101010")*

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

*FileClose (OutputFileHandle)*

## П.2.10. 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*.

*Учебное издание*

**Бахтизин Вячеслав Вениаминович**  
**Куликов Святослав Святославович**  
**Фадеева Елена Павловна**

***АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ ПРОГРАММНОГО  
ОБЕСПЕЧЕНИЯ. РАБОТА В СРЕДЕ SilkTest 8.0***

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор Н. В. Гриневич  
Корректор Е. Н. Батурчик  
Компьютерная верстка Ю. Ч. Ключкевич

Подписано в печать 26.06.2012. Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».  
Отпечатано на ризографе. Усл. печ. л. 4,3. Уч.-изд. л. 4,0. Тираж 100 экз. Заказ 50.

Издатель и полиграфическое исполнение: учреждение образования  
«Белорусский государственный университет информатики и радиоэлектроники»  
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.  
220013, Минск, П. Бровки, 6