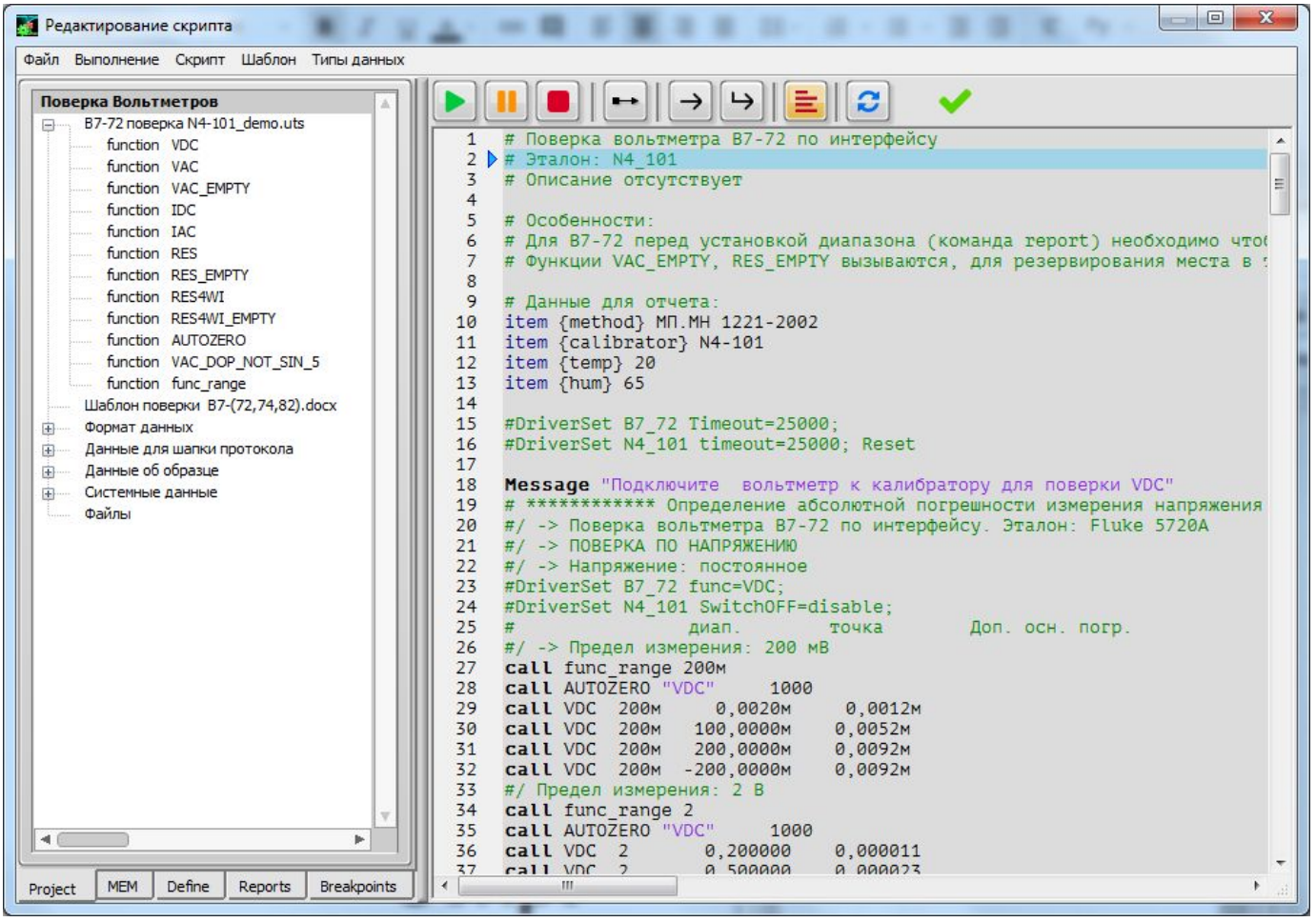
UniTesS



Script

руководство по программированию

версия документа 5.4

Минск

2017

Термины и определения

Разработчик UniTesS – сотрудник, обеспечивающий разработку автоматизированного рабочего места с использованием ПО UniTesS и выполняющий следующие работы: разработка *скрипта*, подготовка *шаблона протокола* и данных для шапки, описание *типа данных для протокола*.
Скрипт (англ. *Script* - сценарий) – файл, в котором в текстовой форме содержится последовательность действий, реализующих методику измерений. Скрипты разрабатываются с использованием языка программирования *UniTesS Script*. Скрипт позволяет реализовать: управление приборами, математические расчеты, сравнения, отправку данных в отчет и т.д.
UniTesS Script – упрощенный язык программирования, предназначенный для реализации методик поверки, калибровки СИ и испытаний оборудования.
Комментарии – *фрагменты текста,* фразы, описывающие участки скрипта, и предназначенные для пояснений. Комментарии всегда начинаются с символа «**#**».
Специальные комментарии – комментарии, предназначенные для формирования удобного представления скрипта в режиме *Simple View*. Символы: **#F/** , **#/** .
Типы данных для протокола – определяют какие именно данные будут заноситься в протокол, а также их формат, подразделяются на категории для различных видов работ.
Данные для шапки протокола – стандартизированные данные, автоматически подставляемые в каждый протокол, такие как: номер протокола, модель устройства, серийный номер, температура, влажность, параметры питающей сети и т.д.
Функции – это участки кода в скрипте, неоднократно вызываемые для исполнения. Функция описывает метод для измерения одного параметра и должна сопровождаться *специальным комментарием* для отображения в режиме *Simple View.*
Шаблон протокола – документ MS Word в формате doc или docx, содержащий закладки (bookmark), созданные по определенным правилам, для автоматического заполнения.
Alias – псевдоним, имя, принятое внутри скрипта.
EOS (англ. end of sting) – *специальный* символ, признак конца строки. Применяется при работе с приборами по интерфейсам для обозначения конца отправленной команды.
VISA интерфейс – широко используемый стандартизированный интерфейс ввода/вывода в области тестирования и измерений для управления приборами. Поддерживает интерфейсы IEEE-488 (GPIB/КОП), PXI, VXI, RS-232, RS-485, USB, Ethernet.
IVI драйвер – архитектура драйверов, разработанная IVI-сообществом, призванная стандартизировать взаимодействие с измерительными приборами, управляемыми по интерфейсу. Использование данной архитектуры обеспечивает взаимозаменяемость измерительных приборов внутри одного класса. В *скрипте* приборы одного класса управляются одинаковыми командами.
Simple View – упрощенный режим просмотра скрипта, в котором пользователь видит наглядную древовидную структуру вызовов функций с параметрами и результат их

выполнения. Для формирования структуры в скрипте применяются *специальные комментарии*.

Timeout – время ожидания отклика по интерфейсу. Параметр связан с отправкой запросов прибору и предназначен для оценки ее выполнения за определенное время. Если прибор не отвечает на запрос в течении времени ожидания, то генерируется ошибка интерфейса.

UniTesS драйвер – специализированный драйвер для использования в ПО UniTesS APM, применяется если для определенного прибора не существует IVI драйвера и управление командами через *VISA интерфейс* усложнено.

UniTesS плагин – программа, которая обеспечивает полную имитацию работы реального прибора, управляется аналогичным набором команд и на выходе выдает определенный набор параметров, виртуальный прибор.

Введение

Данное руководство адресовано *разработчикам ПО UniTesS APM* и содержит информацию, необходимую для создания автоматизированного рабочего места (далее - APM) и настройки базы данных UniTesS DB.

Структурно UniTesS состоит из базы данных UniTesS DB, развернутой на сервере предприятия, клиентского ПО UniTesS Manager и ПО автоматизированного рабочего места UniTesS APM с опциональным модулем машинного зрения Vision.

UniTesS DB содержит всю информацию о выполненных работах, структуре организации, всех подразделениях и сотрудниках, собственных эталонных средствах измерения, документах, заказчиках и т.д.
UniTesS Manager обеспечивает доступ сотрудников к базе данных в зависимости от предоставленных полномочий, предназначено для организации автоматизированного документооборота, контроля за выполнением работ и анализа результатов деятельности лаборатории.
UniTesS APM предназначен для управления приборами, автоматизации измерений выполнения математических расчетов и формирования протоколов.

1. Почему UniTesS Script

Уже много лет мы занимаемся разработкой автоматизированных рабочих мест и аппаратно-программных комплексов для испытательных, поверочных и калибровочных лабораторий. Позади сотни успешных внедрений, в том числе для национальных эталонов РБ. Диапазон реализованных проектов очень широк - от самых простых, таких как поверка манометров или мультиметров, до уникальных, не имеющих аналогов, например испытания абонентских устройств систем ЭРА-ГЛОНАСС и ECall.

На современном рынке доступны решения по автоматизации от известных брендов с солидной историей и большим опытом работы. Начиная разработку своей собственной системы, мы ориентировались на продукты таких брендов NI Labview, NI TestStand, Fluke Metcal, Transmille ProCal, детально анализировали все их особенности, достоинства и недостатки. Также на первых этапах были опробованы различные средства разработки ПО, мы старались систематизировать и стандартизировать процесс разработки таким образом, чтобы максимально его ускорить и минимизировать затраты, гарантируя при этом высокое качество результата. Но в ходе решения поставленных задач возник ряд проблем, не позволявших предоставлять пользователям готовые программные продукты с приемлемым уровнем качества и функциональности за минимально возможные средства и сроки.

Основные проблемы, с которыми пришлось столкнуться в процессе использования готовых программных продуктов и способы их решения, реализованные при разработке UniTesS APM можно свести в таблицу:

Проблема	Решение
Длительный процесс отладки программного продукта, особенно на заключительных стадиях его создания, занимающий до 80% общего времени на разработку.	Отладка работы скрипта и любое изменение кода происходит непосредственно в ходе его выполнения, без потери данных и разрыва сессий. Нет необходимости тратить время на компиляцию и перезапуск программы после корректировки каждой команды или значения одного параметра.
Отсутствие строгой стандартизации программного кода.	Жесткая стандартизация программного кода облегчает “читаемость” и понимание его любым программистом, даже не являющимся его автором. Каждая строка в теле скрипта представляет собой одну логическую операцию, которая является частью

	алгоритма поверки и шаг за шагом реализует методику измерений.
Сложность специфики метрологии для программистов.	Простота языка позволяет привлекать к разработке скриптов для поверки новых приборов специалистов заказчика, инженерный персонал испытательных лабораторий. Создать полноценный функциональный скрипт, реализующий достаточно сложный алгоритм и методику поверки, можно с небольшим набором команд.
Сложность в проведении метрологической аттестации ПО.	Сотни успешных внедрений, в том числе для национальных эталонов РБ.
Короткий жизненный цикл ПО из-за замены эталонов, поверяемых СИ, регулярных изменений в методиках поверки.	Огромная, постоянно растущая, коллекция драйверов для управления целыми классами приборов с одинаковыми наборами команд. Каждое изменение в методиках, средствах измерения, обновление или даже полная замена эталонов, потребует лишь незначительных корректировок в теле скрипта.
Несовместимость различных АРМ, отличия в форматах данных, протоколов, интерфейсах пользователя, отсутствие комплексного подхода к автоматизации в лаборатории.	Наглядный, интуитивно понятный интерфейс конечного пользователя. Постоянный контроль за состоянием процесса и всех переменных, которые участвуют в вычислениях, в том числе - удобная, детальная визуализация.
Опасность повреждения дорогостоящего оборудования, сложность управления приборами.	Возможность поверки приборов, не обладающих интерфейсом взаимодействия с ПК. Использование драйверов и плагинов для эмуляции реального прибора.

Методом проб и ошибок удалось найти решение всех вышеописанных проблем, мы спроектировали оригинальную среду разработки автоматизированных рабочих мест UniTesS со своим языком программирования и солидным набором сервисных и вспомогательных средств.

Язык UniTesS Script - это компактный, простой, но достаточно мощный и специализированный инструмент для комплексной автоматизации любых действий и

расчетов, связанных с испытаниями различных приборов и сложных измерительных устройств. Ознакомившись с набором команд и правилами создания скриптов, поверитель может самостоятельно запрограммировать методику испытаний любого прибора, не обращаясь к программистам, причем сделать это быстро, надежно и эффективно.

Книга познакомит вас с языком UniTesS Script и научит применять его в UniTesS APM – рабочем месте поверителя и испытателя.

2. Как работает UniTesS APM

Автоматизированное Рабочее Место UniTesS APM предназначено для автоматизации процесса поверки и калибровки, испытаний средств измерения. Для полноценного функционирования комплекса UniTesS APM необходимо разработать скрипт, настроить и добавить необходимую информацию в базу данных. Создание автоматизированных рабочих мест, как правило, состоит из следующих этапов:

- Определение перечня используемого оборудования, ***видов работ***. От этого зависит набор драйверов и интерфейсов, с которыми придется работать;
- Анализ методики измерений и определение ***типов данных для протокола***;
- Подготовка ***шаблона протокола*** в текстовом процессоре MS Word;
- Определение ***данных*** *для шапки протокола* (опционально);
- Разработка ***скрипта***;
- Настройка базы данных UniTesS DB.

UniTesS APM представляет собой интерпретатор, который производит построчный анализ, обработку и выполнение исходного кода программы (скрипта). Основная идея интерпретатора - дать возможность реализовать весь алгоритм процесса поверки и испытания прибора с помощью минимального набора операторов языка. В основном, последовательность действий поверителя состоит из ряда настроек приборов, получения информации, некоторого анализа и математических расчетов. Как правило, завершается логическим выводом о соответствии оборудования и оформлением итогового протокола. Суть работы UniTesS APM заключается в том, что аналогичную последовательность действий, описанную скриптом, выполняет программа.

UniTesS APM предлагает два интерфейса пользователя:

- **Simple View** – упрощенный наглядный режим контроля над процессом поверки, реализованном в скрипте, с удобным представлением для поверителя в виде дерева, без функций редактирования;
- **Editor** – режим редактирования и отладки скрипта для *разработчиков UniTesS.*

На рис. 2.1. показан режим редактирования (вызывается нажатием клавиш <**Ctrl + E**>). Слева находится окно со служебной информацией о текущем проекте, а справа – исходный код скрипта. Прямо над ним, в верхней части окна редактора, расположена панель с инструментами управления и отладки.

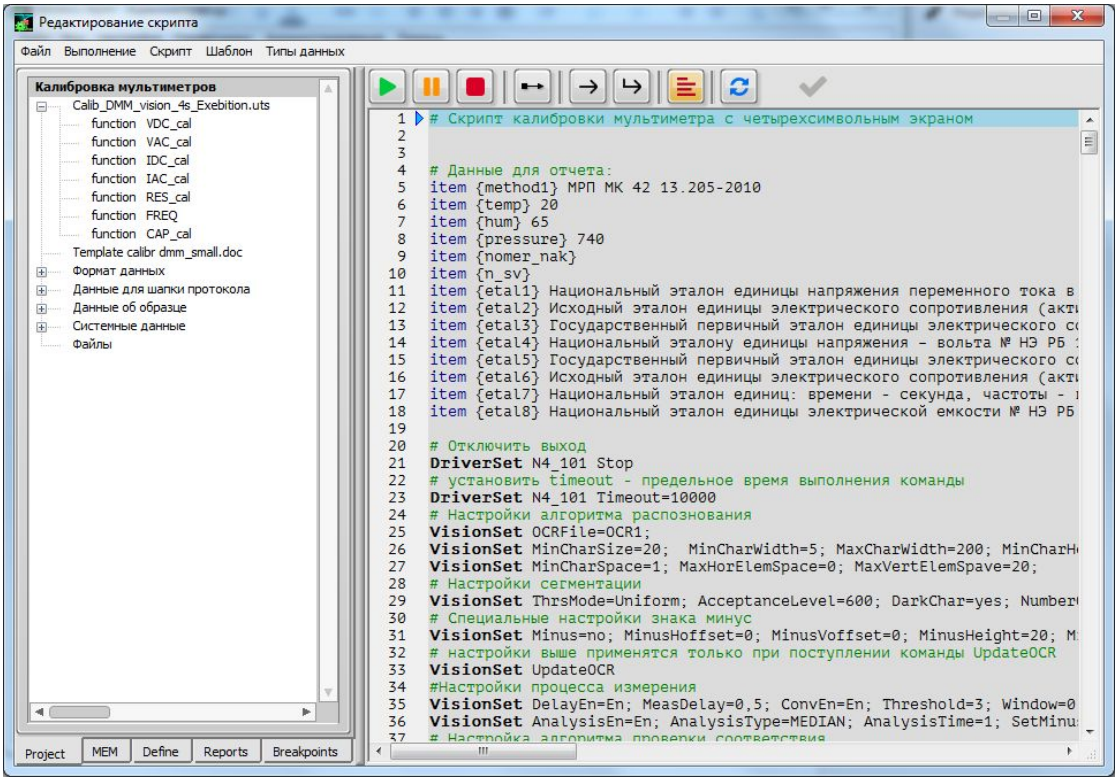


Рис.2.1. Режим редактирования скрипта

Скрипт описывает и задает весь алгоритм действий при выполнении измерений. На этапе создания скрипта разработчик рассматривает и реализует возможности использования встроенных в ПО математических функций, операторов сравнения, построения циклов, подготовки и форматирования данных для отчета, взаимодействия с пользователем, работу с драйверами, а также иные особенности языка. Создать полноценный функциональный скрипт, реализующий достаточно сложную методику поверки, можно с использованием сравнительно небольшого набора команд.

Режим просмотра **Simple View** отображается в главном окне программы UniTesS APM (Рис. 2.2.) и дает наглядное представление о процессе измерения с таблицей конечных результатов, которые заносятся в протокол после выполнения скрипта. Древовидная структура фиксирует все действия, поверяемые точки и обеспечивает возможность контролировать процесс на каждом шаге, а при необходимости - отключить или повторить отдельные этапы измерения.

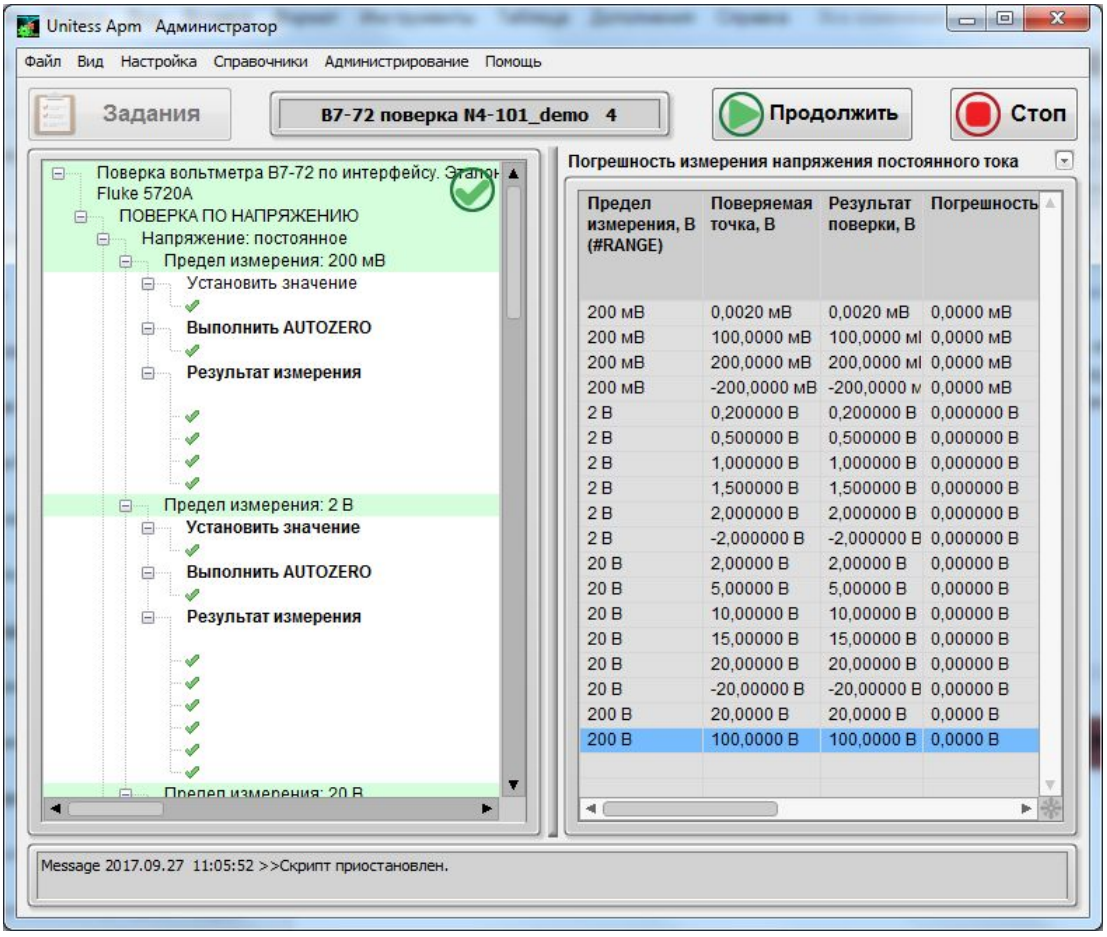


Рис. 2.2. Окно Simple View

Основные функции пользователя-поверителя:

1. Запустить ПО UniTesS APM и авторизоваться;
2. Выбрать задание из списка (или создать его);
3. Подключить оборудование к ПК;
4. Уточнить начальные параметры (если нужно);
5. Нажать “СТАРТ” и следовать инструкциям скрипта (выбор режимов, ввод параметров, коммутация приборов, переключение каналов и т.д.);
6. Подтвердить результаты по точкам, вышедшим за пределы допуска.

Простой интерфейс пользователя позволяет быстро произвести основные настройки и запустить поверку (несколько кликов мыши). При выборе задания из базы данных автоматически загружается **скрипт** для конкретного **вида работ**, **шаблон протокола** и прочие необходимые файлы. После завершения процедуры поверки будет составлен и отправлен в базу данных протокол в формате MS Word и/или PDF. Протоколы формируются на основе готовых шаблонов в формате MS Word и пользователь может легко изменять или добавлять в них любую информацию.

UniTesS APM работает совместно с базой данных UniTesS DB. Все результаты измерений тоже сохраняются в базе. Если процедура поверки не была завершена, то при выборе незавершенной задачи пользователю предлагается Загрузить результаты

или Начать заново. Для нормального функционирования системы необходимо, чтобы данные были доступны на всех автоматизированных местах сотрудникам, которые непосредственно занимаются измерениями, а также руководителям подразделений для распределения заданий и контроля над их выполнением. Поэтому определенным образом производится настройка связей и прав доступа в базе данных.

Для осуществления автоматизации UniTesS APM управляет измерительными приборами, которые подключаются к ПК по интерфейсам: USB, RS232, Ethernet или GPIB, считывает показания, при необходимости выполняет математические расчеты, делает вывод о соответствии и отправляет данные в протокол. UniTesS APM располагает обширной библиотекой драйверов, которая постоянно пополняется, обеспечивает совместимость с широким спектром сложного измерительного оборудования, включая множество классов приборов, управляемых через аппаратно-программный интерфейс National Instruments.

2.1. Перед началом измерений

ПО UniTesS реализует комплексный подход к автоматизации, объединяет такие подсистемы как UniTesS Manager, UniTesS APM и средства работы с базами данных UniTesS DB в единое целое. Приступая к работе с UniTesS APM, перед началом измерений, необходимо выполнить следующие действия:

1. Зарегистрировать все структурные подразделения и пользователей, назначить им уровни доступа, соответствующие служебным обязанностям;
2. Определить **виды работ**;
3. Определить или загрузить в базу данных **типы данных для протокола**;
4. Отредактировать и загрузить в базу данных **шаблоны протоколов**;
5. Загрузить в базу данных набор **скриптов**.

В UniTesS APM предусмотрено несколько видов пользователей, которым можно назначать различные права и полномочия для доступа к базе данных. Обычно пользователи и их разрешения устанавливаются на начальном этапе, при интегрировании системы в лаборатории. В зависимости от делегированных полномочий, им будут доступны различные органы управления и операции. Например, пользователь в должности регистратора может вносить в базу данные о новых средствах измерения, чего не сможет делать инженер, который непосредственно выполняет испытания.

Регистрация пользователей и назначение прав доступа происходит через меню **Администрирование → Организация,** в котором доступны следующие опции:

- Отделы и сотрудники
- Профили доступа
- Должности

Любая лаборатория обычно выполняет строго определенный перечень **видов работ**, их следует систематизировать и четко сформулировать, например:

- испытания на электробезопасность
- испытания по ЭМС
- поверка мультиметров
- поверка осциллографов
- калибровка анализаторов спектра

Типы данных для протокола можно определить в отдельном текстовом файле (имеет расширение .set), который затем загружается в базу данных, либо непосредственно редактировать параметры в окне программы. Файл содержит описание данных, заносимых скриптом в поля шаблона, метод занесения, а также формат записи чисел для каждого поля.

Шаблон протокола - файл в формате MS Word (с расширением .doc или .docx), содержит текст и таблицы, в определенные места добавляются закладки для возможности автоматического заполнения. Закладкам присваиваются имена в соответствии с созданными ранее **типами данных**. Каждой закладке соответствует свой **тип данных**, который и определяет способ их внесения.

Исполняемый **скрипт** - текстовый файл (с расширением .uts), в котором с помощью языка UniTesS Script описана последовательность действий, реализующих методику измерений. Обычно состоит из описания функций, каждая из которых реализует одно конкретное измерение, а также их вызовов, в которых определены контрольные точки и допуски измерений.

Для обеспечения автоматизации рабочих мест внутри каждой лаборатории прежде всего необходимо выполнить подготовку, загрузку, настройку и установить взаимосвязь информации, хранящейся в базе данных. Все эти действия осуществляются из меню “**Администрирование**”, подменю “**Автоматизация измерений**”. Для максимально быстрого старта рекомендуется использовать готовые наборы файлов с **шаблонами**, **типами данных** для протокола и **скриптами**, которые находятся на компакт диске в комплекте поставки ПО. Добавление, редактирование, создание **видов работ**, **шаблонов**, файлов описания **типов данных** и **скриптов** также можно выполнить из меню “**Проекты автоматизации**”.

2.2. Определение видов работ

При настройке базы данных UniTesS DB (Рис. 2.3.) перечень видов выполняемых лабораторией работ следует заполнить в первую очередь, так как все остальные элементы, действия и настройки привязываются к конкретному виду работы.

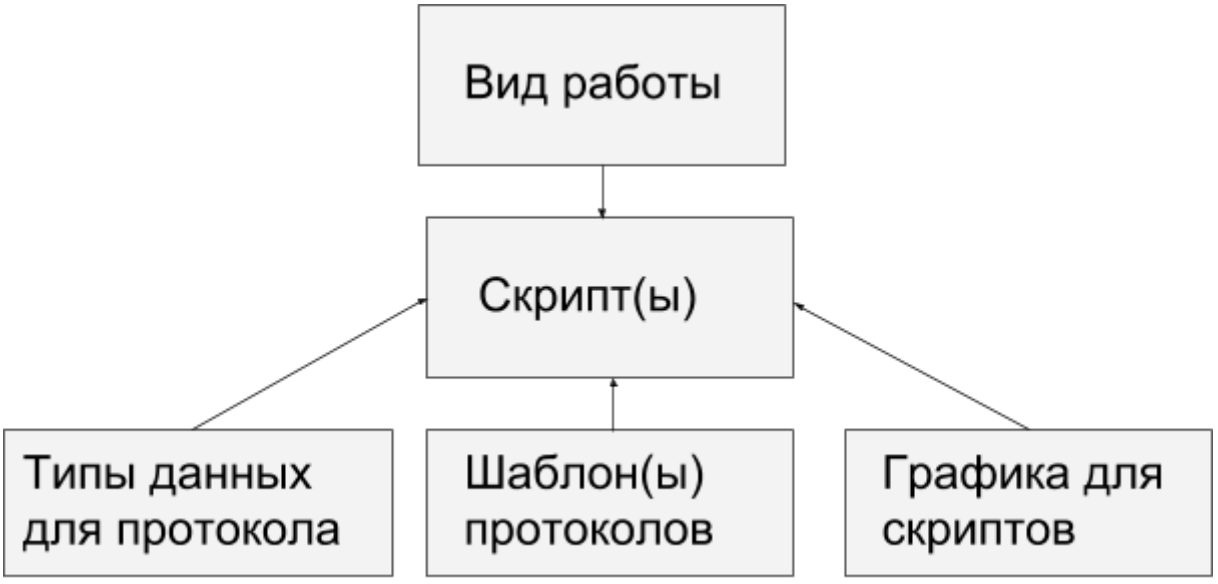


Рис. 2.3. Структура базы данных UniTesS

Для каждого поступающего в лабораторию образца (средства измерения) можно определить один или несколько **видов работ**. По каждому **виду работ** собирается статистика: кто и когда выполнил, дата проверки и подписания протокола, место нахождения образца, формируется отчетность: общее количество поверок, поступление оплаты и т.д.

Открыть перечень **видов работ** можно из меню “**Администрирование**”. При добавлении нового **вида работ** необходимо указать название, типовую стоимость, добавить описание, выбрать сотрудников, допущенных к выполнению, а также этапы (Проверить, Подписать, Утвердить - опционально).

2.3. Определение типов данных для протокола

Типы данных для протокола определяют, какие именно данные и каким образом будут заноситься в **шаблон**. В описании типов содержится информация о форматах данных, по каждому полю в отдельности, а также метод их занесения в **протокол**.

Предусмотрено три метода занесения данных:

- **Table** – заносит данные в текущую строку таблицы, обозначенной закладкой с именем, которое определено в файле;
- **Row** – заносит данные в строку таблицы от места закладки и далее вправо. Предназначен для заполнения отдельных ячеек (идущих подряд) в таблице, а не всей строки таблицы. Применяется для занесения неоднородных данных;
- **String** – заносит строку по месту закладки. Закладка может стоять как внутри таблицы, так и вне ее, в любом месте.

При занесении данных методом **Row** закладка устанавливается на ячейку, с которой начинается ввод данных. Может находиться в любой колонке таблицы, но следует учитывать количество свободных колонок справа от нее для корректного заполнения.

Типы данных для удобства группируются в категории. Каждая категория содержит конкретный набор **типов данных**, определенных для одного класса приборов или **вида работ**. Типы данных можно загрузить из файла (с расширением .set), а затем редактировать в таблице (Рис. 2.4.), в окне программы.

Синтаксис строки файла:

```
data_description <тип данных> <метод> "<Полное название параметра>"; "<название_колонки_1:формат>" ; "<название_колонки_2:формат>" ; … "<название_колонки_n:формат>".
```

где:
<тип данных> - название типа данных, должно совпадать с названием соответствующей метки в файле шаблона протокола;
<метод> - метод занесения в протокол (**Table**, **Row** или **String**);
<Полное название параметра> - название таблицы, помеченной соответствующей меткой в шаблоне протокола (для метода **Table**);
<название_колонки_n:формат> - название колонки в таблице, помеченной соответствующей меткой в шаблоне протокола (для метода **Table**) с описанием формата данных.

Описание формата отделяется от названия колонки или параметра символом двоеточия, каждый элемент берется в кавычки, разделителем служит символ точки с запятой.

Пример строки файла:

```
data_description StepResponseTime table "Время нарастания";"№ Канала:%,;%-6.3r";"Коэффициент отклонения, В/дел.:%,;%-6.3r";"Значение напряжения, В:%,;%-6.3r";"Полярность фронта:%,;%-6.3r";"Допустимое время нарастания, не более, нс:%,;%-6.3r";"Измеренное значение, нс:%,;%-6.3r";"Вывод о соответствии:%,;%-6.3r"
```

Файл позволяет описать формат представления данных в каждой колонке таблицы протокола. Также в файле можно указать формат данных по умолчанию. Он применяется для тех данных, для которых формат не был указан индивидуально.

Примеры строки формата по умолчанию:

```
default_format "%.;%^#_12r"
```

Расшифровка значений:

- Разделителем дробной части является точка;
- Число форматируется в инженерном виде, если используются коды преобразования **e** или **g** в строке форматирования;
- Младшие нулевые разряды удаляются;
- Выравнивание по левому краю;
- Добавляются русские постфиксы в системе СИ.

```
default_format "%,;%.0dw"
```

Расшифровка значений:

- Разделителем дробной части является запятая;
- Постфикс отображается через пробел;
- Десятичный формат, дробная часть опускается.

Ниже приведен набор полей, из которых может состоять строка формата:

```
%[$][-][+][#][^][0][Ширина][.Точность или _ЗначащиеРазряды]Код Преобразования
```

Где *Ширина* должна быть представлена числом больше нуля, а *.Точность* и *_ЗначащиеРазряды* – большим или равным нулю. Нельзя использовать оба модификатора *.Точность* и *_ЗначащиеРазряды* в одной строке.
При использовании в записи с плавающей точкой *.Точность* определяет число разрядов справа от десятичной точки. Если параметр опущен, по умолчанию используется шесть разрядов. Если *Точность* указана как 0, дробная часть будет опущена.

_ЗначащиеРазряды округляют отображаемое число до количества разрядов, указанных пользователем.
Подробное описание элементов формата находится в Приложении № 4.
Примеры спецификаторов форматов приведены в Приложении № 5.

Важно! Название закладки в **шаблоне протокола** обязательно должно совпадать с наименованием **типа данных**. Количество параметров **типа данных** для метода занесения **Table** должно совпадать с количеством колонок итоговой таблицы результатов в **шаблоне протокола**.

Добавление **категории типов данных** выполняется из меню “**Администрирование**” --> “**Автоматизация измерений**” --> “**Типы данных для протокола**”. В окне типов данных нажмите кнопку “Добавить”. Для добавления категории типов данных из файла нажмите кнопку “Загрузить” и выберите нужный файл с описанием типов данных. По завершении всех действий нажмите кнопку “Добавить”.

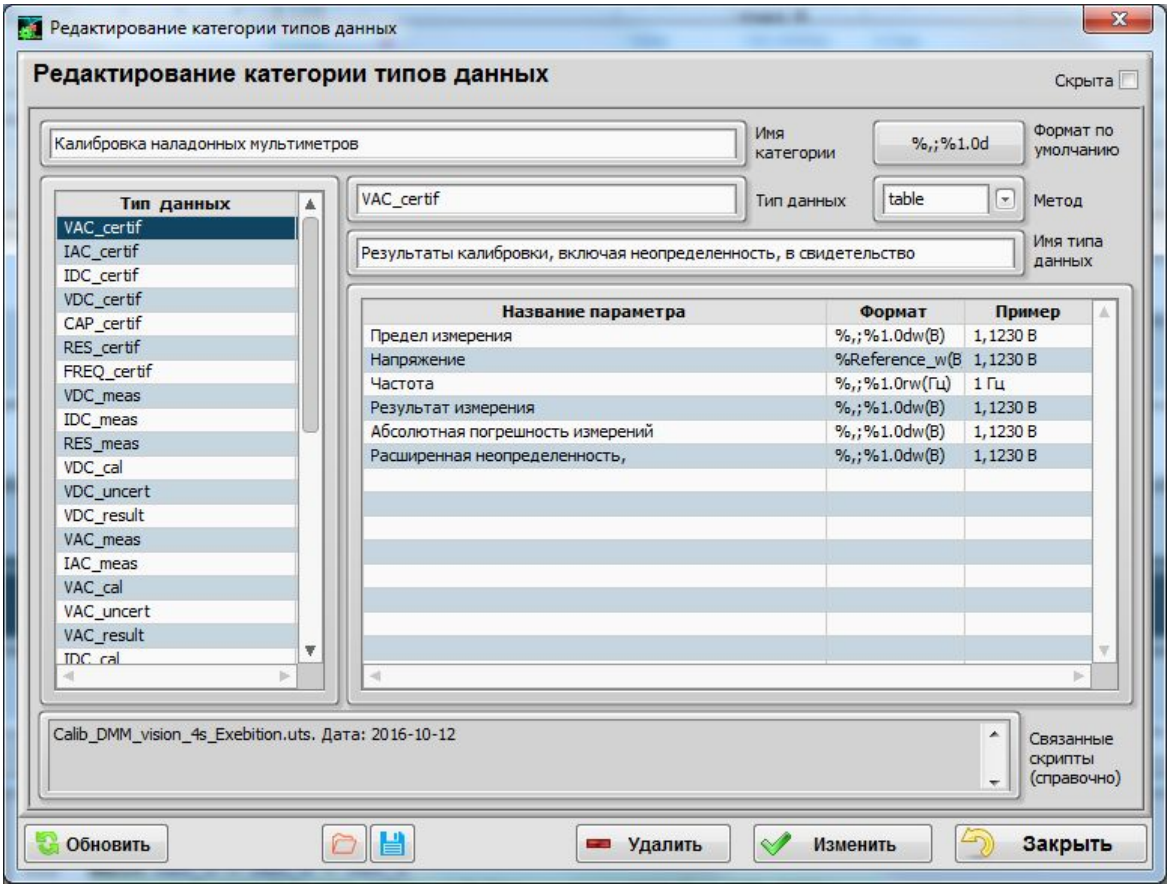


Рис. 2.4. Редактирование категории типов данных.

После загрузки новой категории **типов данных** возможно ее редактирование. Переход в режим редактирования типов данных для протокола (Рис. 2.4.) осуществляется из меню “**Администрирование**” → “**Автоматизация измерений**” → “**Типы данных для протокола**”.

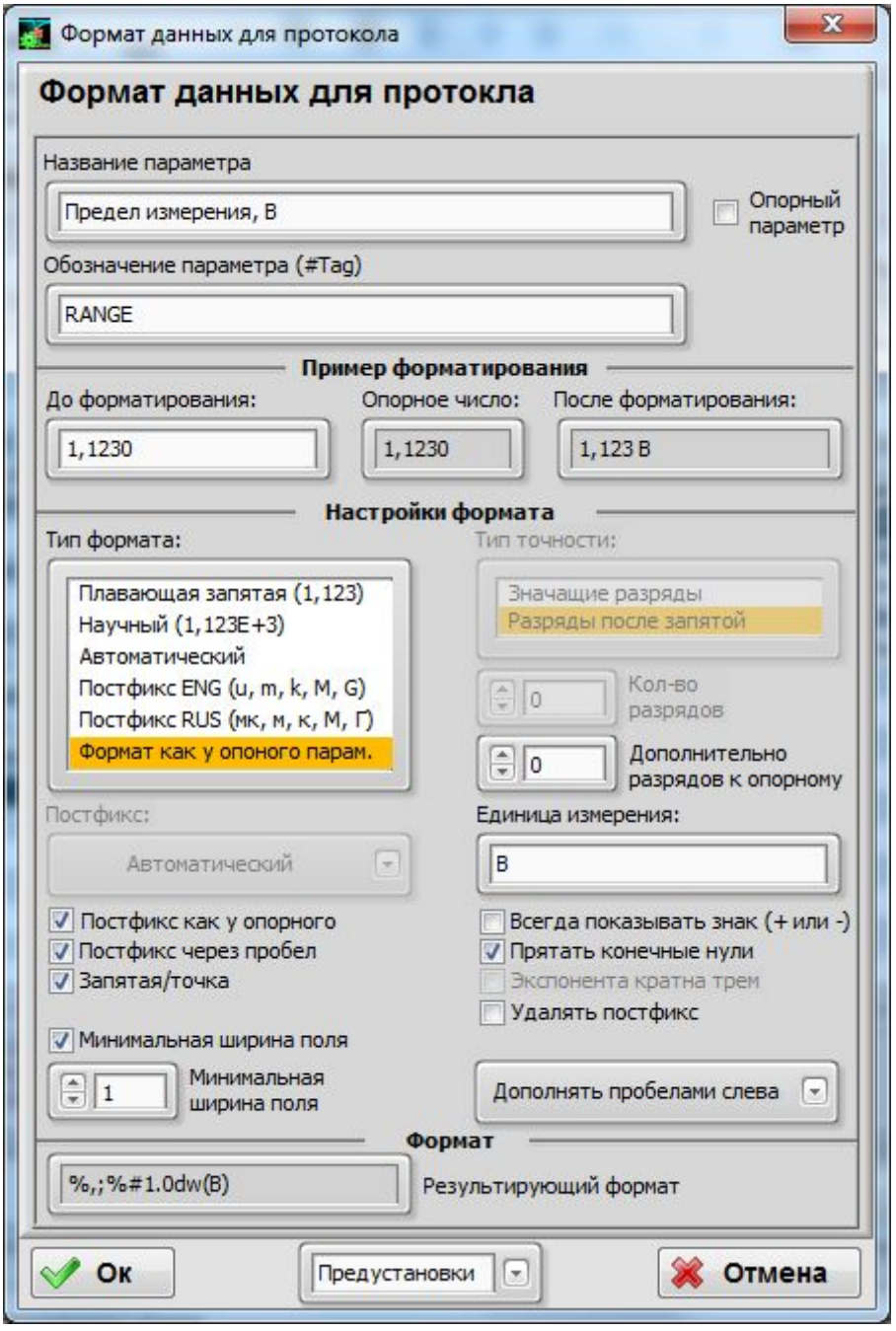


Рис. 2.5. Формат данных для протокола.

При настройке Формата данных для протокола (Рис. 2.5.) для каждого параметра можно указать ряд значений:

- название параметра;
- обозначение параметра (тег);
- тип формата;
- точность (число значащих разрядов или разрядов после запятой);
- постфикс и его положение;
- единица измерения;
- значение минимальной ширины поля.

В процессе настройки корректность устанавливаемых параметров можно контролировать на **Примере форматирования**. Значения вводятся последовательно, разделителем служит символ точки с запятой.

Если один из форматов выбирается как “**Опорный параметр**” (флажком устанавливается соответствующая опция и результирующий формат выглядит как **%Reference_w**), то описание остальных значительно упрощается - ряд значений автоматически наследуется по шаблону “Как у опорного”.

После того как необходимые **типы данных** загружены в базу, можно перейти к редактированию и загрузке **шаблонов протоколов**.

2.4. Формирование шаблона протокола

Шаблон протокола создается с использованием текстового процессора MS Word. Для упрощения задачи рекомендуется редактировать готовые шаблоны из комплекта поставки АРМ. На Рис. 2.6. приведен пример фрагмента **шаблона протокола** для поверки осциллографов.

НАЗВАНИЕ ПОДРАЗДЕЛЕНИЯ
ОРГАНИЗАЦИЯ

АДРЕС
телефон

ПРОТОКОЛ № D_PROTOCOL_NUM
поверки осциллографа универсального D_MODEL

Заводской номер: D_SERNUM
Нормативные документы по проведению поверки:
{method1}
{method2}
{method3}

Условия поверки:

Температура окр. среды	(20 ± 5) °С	{temp}
Относит. влажность	65 ± 15 %	{hum}
Атмосферное давление	(750 ± 30) мм рт. ст.	{pressure}
Напряжение	(220±4,4) В	{voltage}
Частота	(50±0,5) Гц	{frequency}

Средства поверки

Наименование, тип, заводской номер	Метрологические характеристики	
Калибратор осциллографов импульсный И1-9 № 0908	3 мкВ - 100 В, 100нс - 10с	ПГ ±(2,5·10^{-3}Uг +3мкВ ПГ ± 10^{-4}t
Генератор испытательных импульсов И1-15 № 262	0,1 - 10 мкс, 10 В $t_ф \leq 0{,}25$ нс	ПГ ± (0,1τ + 0,01)мкс ПГ ± 10%
Вольтметр универсальный цифровой GDM-8245 № E110130	U= 10^{-5} - 1200 В U~ 10^{-5} - 1000 В 20 Гц - 50 кГц	ПГ ± (0,0003 $U_{изм}$+4 ЕМР) ПГ ± [(0,005 - 0,05) $U_{изм}$ + (15 - 30) ЕМР]

Результаты поверки:
1. Внешний осмотр: {ext_test} требованиям МП
2. Опробование: {testing} требованиям МП
3. Результат самодиагностики:

4. Определение метрологических характеристик

Таблица 4.1 Определение погрешности коэффициента отклонения

№ Канала	Коэффициент отклонения, В/дел	Значение напряжения, В	Нижний предел допускаемой абс. погрешности коэффициента отклонения, В	Верхний предел допускаемой абс. погрешности коэффициента отклонения, В	Измеренное значение амплитуды, В	Вывод о соответствии

Таблица 4.2 Определение времени нарастания переходной характеристики

№ Канала	Коэффициент отклонения, В/дел	Значение напряжения, В	Полярность фронта	Допустимое время нарастания, не более, нс	Измеренное значение времени нарастания, нс	Вывод о соответствии

Рис. 2.6. Пример шаблона протокола.

Шаблон протокола можно условно разделить на несколько логических частей.

Первая логическая часть состоит из заголовка документа и стандартных сведений о процедуре поверки (условия проведения поверки - температура, давление, влажность, модель прибора, серийный номер, наименование лаборатории, фамилия специалиста, проводившего поверку, эталонное оборудование и т.п.). Все эти данные берутся из базы данных или вводятся пользователем на начальном этапе, заносятся в протокол автоматически.

Вторая логическая часть содержит результаты поверки - точные данные о показаниях и погрешностях измерений прибора на всех контрольных точках в виде таблиц или отдельных полей. Это информация, получаемая в процессе работы скрипта по результатам измерений, она заносится в протокол специальной командой **Report**.

Финальная часть протокола, Заключение о поверке, как и заголовок, может иметь произвольную форму и содержать любые настраиваемые поля. При составлении заключения UniTesS APM использует зарезервированные переменные:
{name} – Ф.И.О. поверителя;
{position} – должность поверителя;
{test_res} – результат поверки;
{date} – дата проведения поверки.

Существует предопределенный набор системных полей с псевдонимами:
{testing_result} - Глобальный результат выполнения скрипта. Вместо этого псевдонима UniTesS APM подставит фразу о положительном или отрицательном результате тестирования, причем фраза будет взята для конкретного отдела текущего сотрудника из таблицы S_APPROVED;
{user_full_name} - Фамилия И.О. текущего пользователя из таблицы SYS_USERS;
{user_position} - Полное название должности для текущего пользователя из таблицы S_JOB;
{user_department} - Полное название отдела для текущего пользователя из таблицы S_DEPARTMENT;
{current_time} - Текущая дата и время в формате в соответствии с настройками операционной системы;
{start_time} - Время начала тестирования в формате в соответствии с настройками операционной системы;
{sw_ver_apm} - Текущая версия ПО UniTesS APM;
{sw_ver_db} - Текущая версия базы данных;
{scr_name} - Имя выполняемого скрипта;
{scr_date} - Дата последней модификации текущего скрипта в базе данных;
{scr_ver} - Версия скрипта, берется из текста скрипта из команды ScriptVersion;
{scr_uhesh} - Контрольная сумма скрипта. Рассчитывается без учета пробелов, табуляций, символов переноса строки, комментариев, команд Item.

Значения зарезервированных закладок {name} и {position} берутся из персональных данных поверителя, который успешно авторизовался, то есть ввел свой пароль при загрузке UniTesS АРМ.

Следует обратить внимание, что:
- значения зарезервированных переменных, за исключением {test_res}, впоследствии изменить нельзя. В зависимости от результата поверки закладка {test_res} принимает одно из двух значений, задаваемых в текстовых полях “Итоговый положительный результат” и “Итоговый отрицательный результат”, которые хранятся в базе данных;
- одна переменная в шаблоне протокола может использоваться любое количество раз.

Если при внесении данных в протокол возникает необходимость переноса информации на новую строку, то в качестве символа перевода строки используется **\n**. Например:

```
"перенос \n строки"
```

В **шаблоне протокола** также можно использовать набор зарезервированных стандартных параметров, которые хранятся в базе данных UniTesS. Это Данные о средстве измерения и Системные данные. Ключевые слова начинаются с префикса "**D_**". Все параметры, кроме данных команды **Report**, автоматически заносятся в протокол с помощью стандартной функции текстового процессора Word "найти и заменить".

D_PROTOCOL_NUM - номер протокола;
D_MODEL - модель прибора;
D_CLASS - класс СИ;
D_GOSS_REESTR - номер в госреестре;
D_SERNUM - идентификатор, серийный номер прибора;
D_ZAIVKA - номер заявки;
D_WORK_NAME - вид работы;
D_ID_DPRT - идентификатор вида работы для образца;
D_ID_DUT - идентификатор образца;
D_ORG_ZK_NAME - наименование организации заказчика;
D_ORG_ZK_ADDRESS - адрес организации заказчика;
D_USER_JOB - должность специалиста, проводившего поверку;
D_USER_FIO - фамилия, имя, отчество специалиста, проводившего поверку;
D_CURRENT_DATE - текущая дата;
D_REG_DATE - дата регистрации заявки.

Регистр символов в названиях этих параметров не критичен. Все варианты написания

```
D_PROTOCOL_NUM
d_protocol_num
D_Protocol_Num
```

корректны.

При формировании итогового файла протокола программа ищет эти названия и подставляет необходимые значения. Список ключевых слов и их значений для каждого конкретного скрипта можно посмотреть в редакторе **скрипта** (закладка **Projects**), в разделах “Данные для шапки протокола”, “Данные об образце” и “Системные данные”. Таким образом можно импортировать в шаблон любые сведения о средствах измерения, заказчиках из базы данных или создавать нужные поля автоматического заполнения самостоятельно. Значения, которые связаны с условиями текущей проверки, пользователь вводит при запуске скрипта. Для удобства пользователя существует возможность копирования названий ключевых слов (при помощи правой клавиши мыши) и корректной вставки в шаблон.

Неотъемлемой частью Протокола являются колонтитулы, они печатаются на каждой странице, вверху либо внизу, и могут содержать, например, информацию с названием организации, подразделения, контактные данные.

Добавление в базу данных нового шаблона протокола осуществляется из меню “**Администрирование**” --> “**Автоматизация измерений**” --> “**Шаблоны протоколов**”. В окне шаблонов протоколов (Рис. 2.7.) нажмите кнопку “Добавить”. В окне добавления шаблона протокола необходимо выбрать нужный файл (в формате MS Word с расширением .doc или .docx), заполнить поля: “Описание”, “Данные для отчета” (опционально).

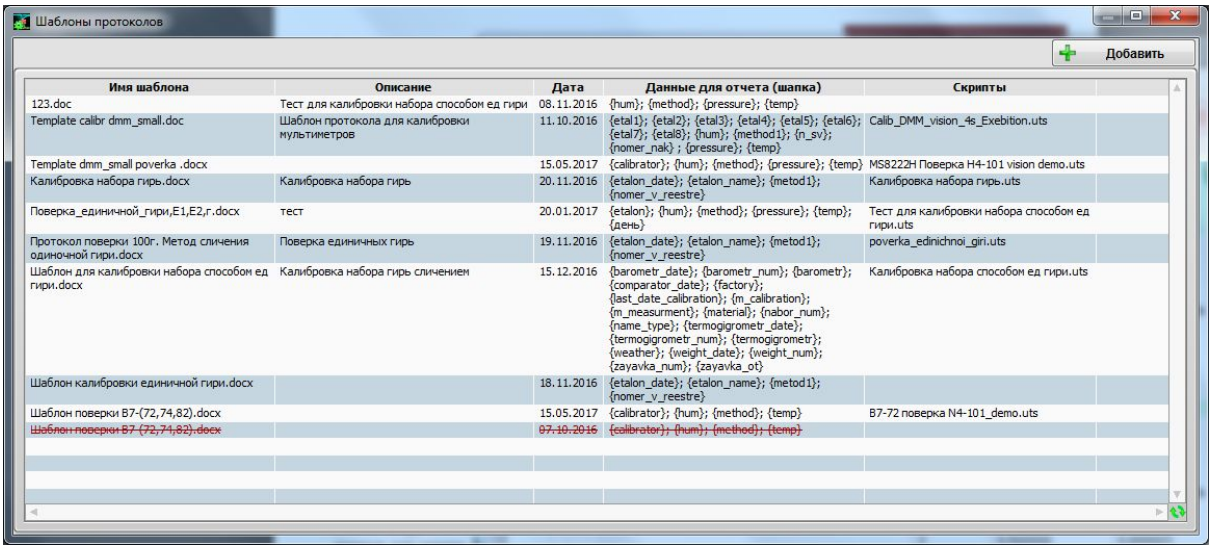


Рис. 2.7. Загрузка шаблона протокола.

Для каждого шаблона можно указать поля данных для отчета (данные для шапки протокола - это набор параметров, которые заполняются пользователем перед началом испытаний). Их также можно загрузить из файла, а затем редактировать вручную в таблице. Данный этап не является обязательным, так как все необходимые для формирования отчета данные можно получить командами скрипта в процессе его выполнения.

Шапка протокола содержит заголовок (Название) и описательную часть, набор форматизированных выражений (Тег). Заголовок может содержать любые сведения, информацию на русском языке. Здесь же описываются условия проведения измерений. Программное обеспечение заполняет заголовок в шаблоне с использованием переменных (например, Серийный номер - {sn}). Размер поля описательной части ограничен тридцатью символами. Тег всегда заключается в фигурные скобки и состоит из набора латинских букв и цифр, может включать символ подчеркивания или тире.

При составлении протокола ПО UniTesS находит переменную {sn} и заменяет ее значением, введенным поверителем в одно из полей “Данные для отчета” (Рис. 2.8.):

Название	Тег
Номер в реестре	{nomer_v_reestre}
Наименование ГОСТ	{metod1}
Наименование эталона	{etalon_name}
Дата следующей поверки эталона	{etalon_date}
Серийный номер	{sn}

Рис. 2.8. Добавление данных для отчета.

Выбор нужного протокола для редактирования осуществляется из меню “**Администрирование**” → “**Автоматизация измерений**” → “**Шаблоны протоколов**”.

Порядок добавления дополнительных сведений:

1). В шаблоне протокола укажите новую строку, например:
Давление: {pressure}
где {pressure} – произвольная уникальная переменная, может использоваться любой набор символов.

Важно: не используйте переменные со следующими уникальными именами: {name}, {position}, {test_res}, {date}. Они являются зарезервированными.

2). В окне “Редактирование информации о шаблоне” (Рис. 2.9.) в таблице “Данные для шапки протокола” необходимо добавить новую строку:

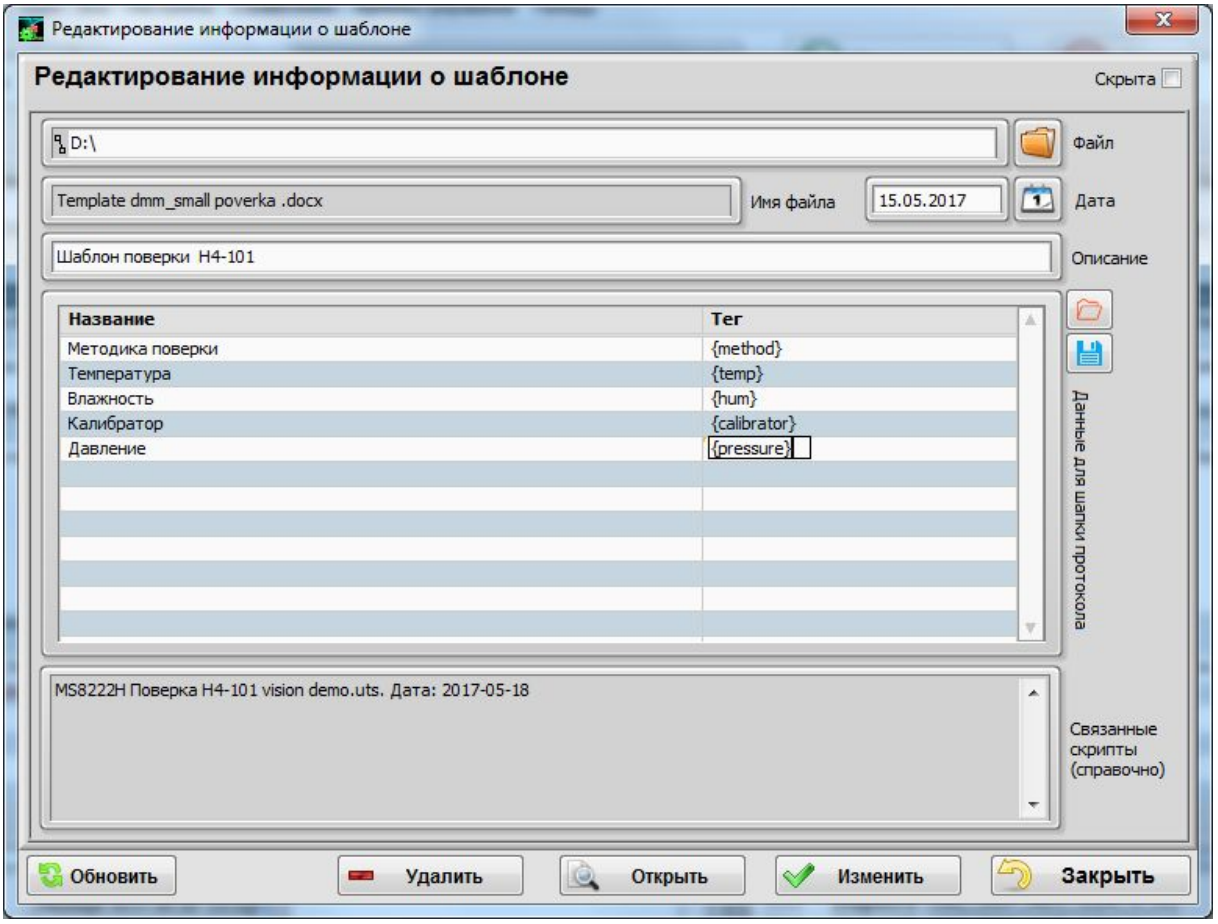


Рис. 2.9. Редактирование информации о шаблоне.

Укажите название, которое будет отображаться для поверителя в UniTesS APM и соответствующий ему тег, например: Давление - {pressure}.
Эти действия можно выполнить из соответствующего меню ПО UniTesS APM или UniTesS Manager.

3). Сохраните изменения и при следующем запуске поверки в ПО UniTesS в поле “Данные для шапки протокола” (Рис. 2.10.) появится новая запись:

Название	Тег
Методика поверки	{method}
Температура	{temp}
Влажность	{hum}
Калибратор	{calibrator}
Давление	{pressure}

Рис. 2.10. Данные для шапки протокола.

При составление протокола о поверке ПО UniTesS автоматически заменит переменную {pressure} на значение, указанное поверителем.

Примечание. Этап заполнения данных для шапки протокола не является обязательным, описание приведено для совместимости с ранними версиями ПО. Полный набор данных для отчета можно сформировать командами скрипта в процессе его выполнения.

Создание отчета происходит на заключительном этапе, после завершения выполнения скрипта. Занесение данных в отчет осуществляется командой **Report** по месту закладок. Необходимо помнить, что пустые таблицы в шаблоне отчета удалять не следует, даже если они не используются при текущей поверке. Если прибор не проходит поверку по какому-либо параметру, ПО автоматически удалит пустую неиспользуемую таблицу из протокола.

Порядок столбцов в таблице шаблона должен совпадать с порядком следования данных в строке скрипта с вызовом команды **Report**. ПО UniTesS заполнит таблицу в таком же порядке, который указан в скрипте.

Чтобы избежать размещения названия таблицы и самой таблицы на разных страницах итогового документа, рекомендуется в параметрах таблицы файла MS Word запретить разбиение на страницы.

Каждая таблица **шаблона протокола** помечается специальной закладкой. Имя закладки соответствует названию одного из типов данных. Это необходимо для того, чтобы программа могла распознать нужные таблицы и корректно произвести их заполнение в отчете. Если таблице не присвоена закладка – ее заполнение не произойдет, она останется пустой. Каждая таблица должна содержать строго определенное число столбцов. Важно помнить об этом при создании своего собственного шаблона “с нуля”.

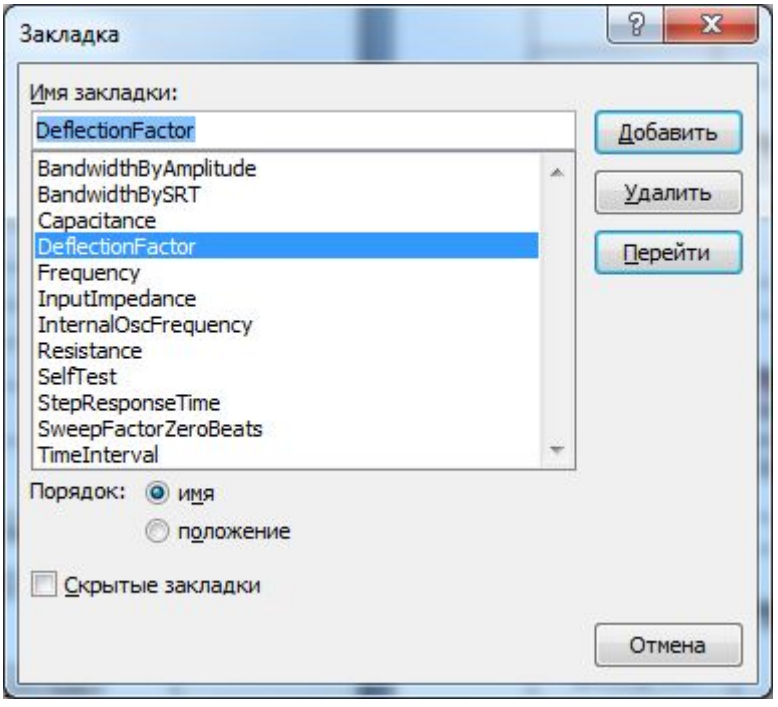


Рис. 2.11. Определение закладки в MS Word.

В MS Word закладки находятся в меню “Вставка” (Рис. 2.11.). Для охвата закладкой всей таблицы необходимо выделить таблицу целиком, нажав на символ над левым верхним углом (Рис. 2.12.).

Таблица 2.1 Определение

№ Канала	Коэффициент отклонения, В/дел	Значе напряже
1	100м	600

Рис. 2.12. Выделение всей таблицы в MS Word.

Затем в меню “Вставка” нажать кнопку “Закладка”, в открывшемся окне присвоить имя закладке и нажать ”Добавить”. После заполнения данными шаблона программа удалит все пустые таблицы, причем удалено будет все, что было выделено закладкой. Поэтому, если у таблицы есть название, нужно выделять ее вместе с названием. Параметры, определенные как **типы данных**, являются общими для всего предприятия и имеют единый формат. Если для разного оборудования требуется проверить один параметр, но с различными наборами полей, то следует вводить новый параметр под другим именем.

Для удобства просмотра, чтобы все добавленные закладки отображались на экране при редактировании документа, в Параметрах Word, в секции “Дополнительно” (подраздел “Показывать содержимое документа”) следует установить опцию “Показывать закладки” (Рис. 2.13.).

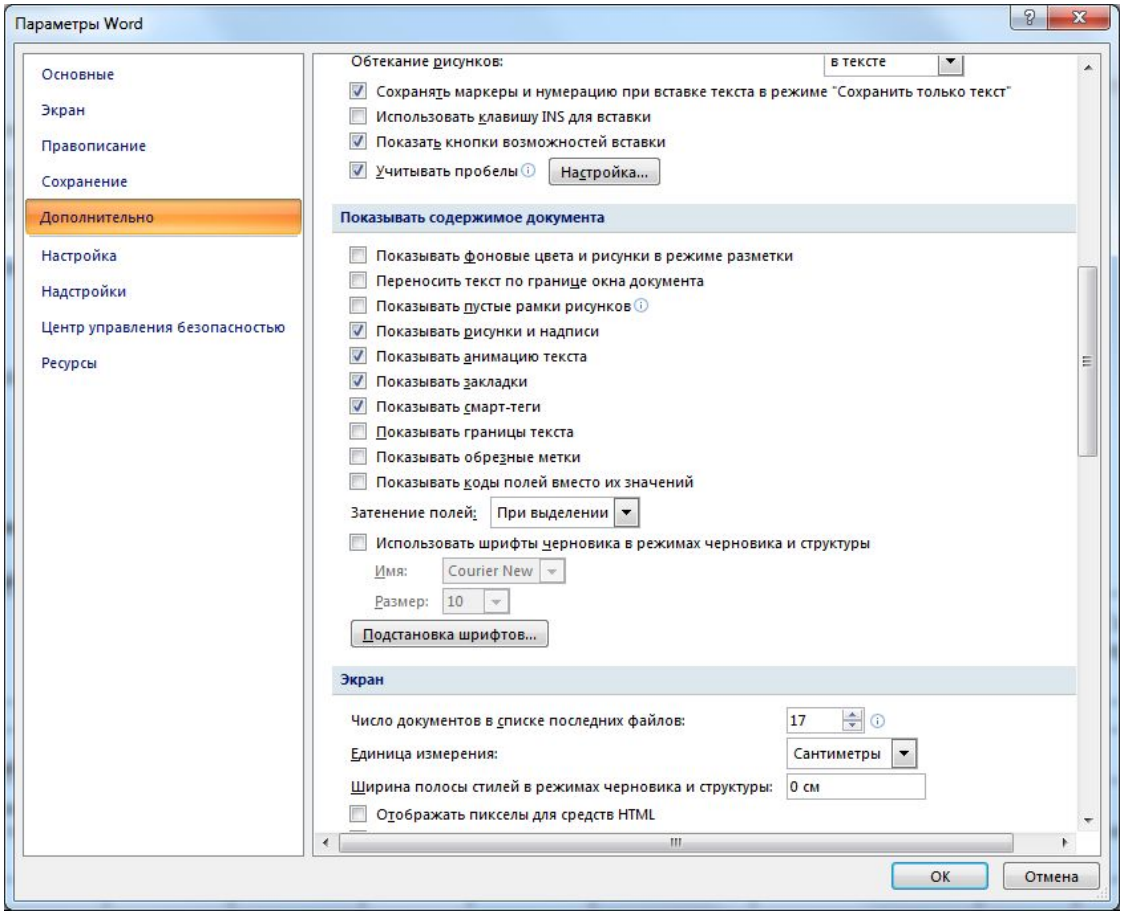


Рис. 2.13. Дополнительные режимы просмотра документа в MS Word.

Каждая закладка в шаблоне протокола при установленном режиме будет помечена квадратными скобками.

Пример заполнения таблицы методом table:

Предположим, что нам необходимо занести данные в таблицу с семью колонками.

Тогда команда **report** будет выглядеть так:

```
report DeflectionFactor mem_1 mem_2 mem_3 mem_4 mem_5 mem_6 mem_7
```

Строка, описывающая тип данных DeflectionFactor:

```
data_description DeflectionFactor table "Коэффициент отклонения";
"№ Канала:%,;%-6.0r"; "К-т отклонения, В/дел.:%,;%-6.3r";"Выходное
напряжение калибратора, В:%,;%-6.3r";"Нижний предел допускаемой
абсолютной погрешности, В:%,;%-6.3r";"Верхний предел допускаемой
абсолютной погрешности, В:%,;%-6.3r";"Измеренное значение
амплитуды, В:%,;%-6.3r";"Вывод о соответствии:%10s"
```

Заполненная строка в таблице протокола, при условии, что вся таблица была выделена закладкой с названием “DeflectionFactor”, будет выглядеть так:

Таблица 2.1 Определение погрешности коэффициента отклонения

№ Канала	Коэффициент отклонения, В/дел	Значение напряжения, В	Нижний предел допускаемой абс. погрешности коэффициента отклонения, В	Верхний предел допускаемой абс. погрешности коэффициента отклонения, В	Измеренное значение амплитуды, В	Вывод о соответствии
1	100м	600м	510м	690м	630м	Соот.

Пример заполнения строки таблицы методом row:

```
data_description Data row "Напряжение";"Контрольная
точка:%,;%-6.3r";"Смещение:%,;%-6.3r";"Амплитуда, В:%,;%-6.3r"
```

Заполненная строка в таблице при условии, что закладка в шаблоне установлена на ячейку в третьем столбце, будет выглядеть так:

		600м	540м	601м		

Пример заполнения отдельной строки методом string:

```
data_description SelfTest string "Самодиагностика прибора";
"Результат: %15.8s"
```

При условии, что в шаблоне установлена соответствующая закладка, заполненное поле выглядит следующим образом:

Самодиагностика прибора
Результат: готов

Шаблон может содержать любое количество закладок, и если в закладку не было произведено записей в конце формирования протокола, то она удаляется. Таким образом, можно использовать один шаблон со множеством скриптов.

Алгоритм заполнения шаблона.
Шаблон заполняется программой MS Word по командам UniTesS APM.
1 Шаг. Файл с шаблоном открывается в фоновом режиме. Если на момент запуска Word уже открыт, пользователю предлагается сохранить данные и закрыть Word.
2 Шаг. Данные форматируются с учетом правил типа данных (data description).

3 Шаг. С помощью функции "найти и заменить" происходит замена всех системных ключевых полей, заносятся данные для шапки протокола и данные об образце.
4 Шаг. Формируется список применяемых в шаблоне закладок. Закладки сортируются на типы string, table, row.
5 Шаг. По всем закладкам, исходя из типа, вносятся данные.
6 Шаг. Незаполненные закладки удаляются.
7 Шаг. Сформированный документ сохраняется под именем номера протокола и открывается пользователю в редакторе Word.

2.5. Загрузка скриптов

В последнюю очередь в базу данных загружаются **скрипты** – файлы содержащие описание методики поверки, калибровки или испытаний в понятной для программы форме. Вид окна редактирования информации о скрипте приведен на рис. 2.15.

Добавление скрипта осуществляется из меню “**Администрирование**” --> “**Автоматизация измерений**” --> “**Скрипты**”.

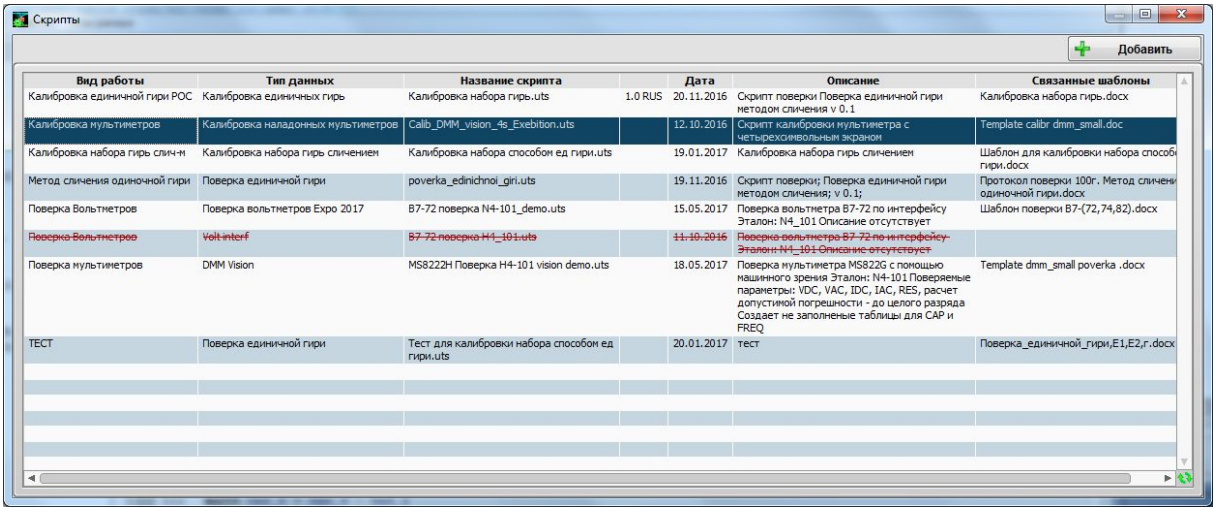


Рис. 2.14. Загрузка скриптов

В окне перечня скриптов (Рис. 2.14.) нажмите кнопку “Добавить”. При добавлении скрипта необходимо выбрать **вид работы**, присвоить категорию **типов данных** (загружена в базу данных ранее), выбрать нужный файл (с расширением .uts), заполнить поле: “Описание” и выбрать **шаблоны**. По завершении всех действий нажмите кнопку “Добавить”.

После загрузки необходимо связать данные между собой (Рис. 2.15.). Каждый **скрипт** привязывается к определенному **виду работы** и категории **типа данных**. Для одного **вида работы** можно назначить несколько **скриптов**, в этом случае пользователю предлагается выбор нужного для текущей поверки. Для каждого **скрипта** назначается один или несколько **шаблонов протокола**, например на русском и английском языке. К **шаблонам**, в свою очередь, могут быть привязаны данные для шапки протокола (опционально).

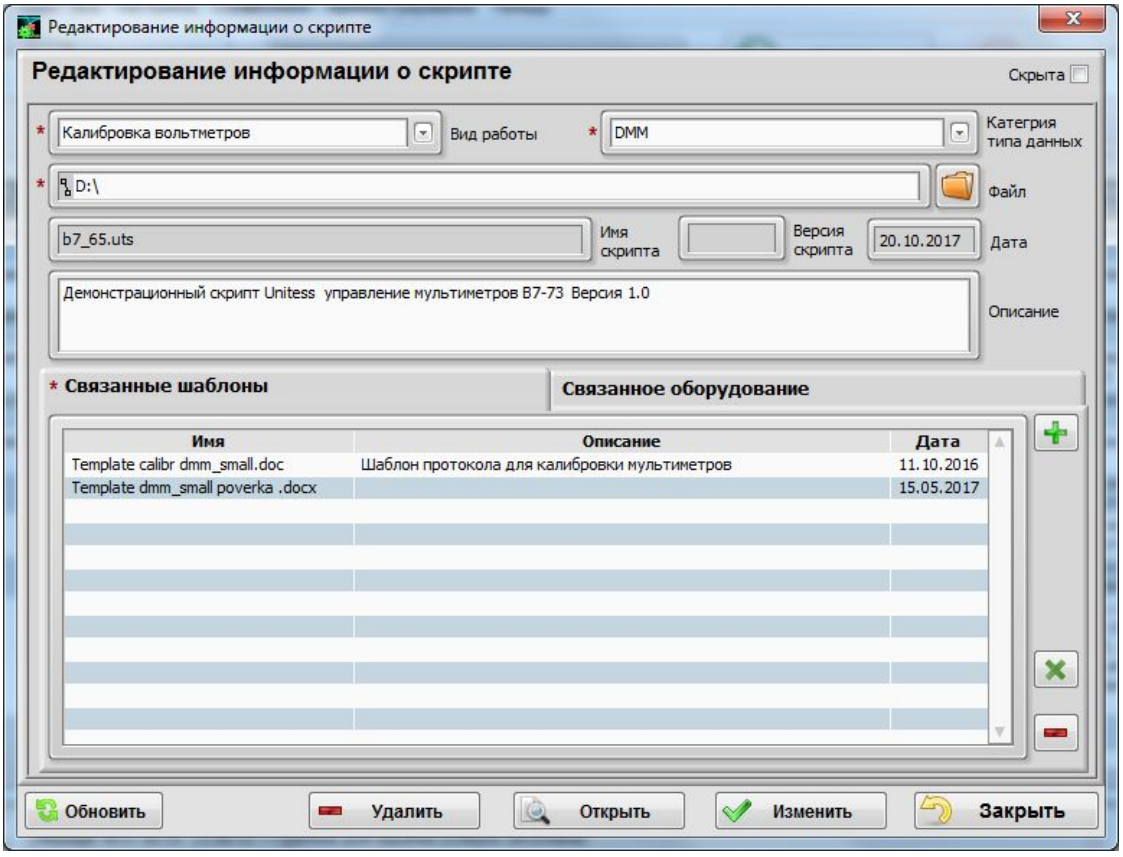


Рис. 2.15. Редактирование информации о скрипте

3. Разработка скрипта

Реализация методики поверки приборов на любом языке программирования высокого уровня неизбежно обернется множеством страниц кода для описания каждого шага, и парой операторов здесь не обойтись. Кроме того, придется изучить гору справочной литературы по методике измерений, взаимодействию с приборами, задействовать море библиотек, фреймворков и прочих составляющих современных систем программирования. Эти обстоятельства значительно повышают расходы на разработку и стоимость создания подобного программного продукта может исчисляться сотнями тысяч долларов. Время разработки растягивается на долгие месяцы и даже годы. Возникает закономерный вопрос: как сократить затраты ресурсов времени, персонала и финансов на разработку и внедрение АРМ, связанного с испытаниями конкретных средств измерения. В процессе создания любого программного обеспечения до 80% времени занимает его отладка, причем в случае необходимости компиляции приходится каждый раз останавливать эксперимент, разрывать сессии и отключать оборудование, а затем повторять весь цикл с самого начала. Некоторые измерения длятся часами, а с учетом замены набора исполняемых файлов на исправление каждой ошибки может быть потрачено несколько дней. Использование UniTesS АРМ под управлением простого языка UniTesS Script позволяет вносить любые корректировки в программный код “на ходу”, без остановки процесса поверки, прерывания сессий и потери данных. Типизированная архитектура скриптов делает программирование легким и приятным. Наглядная древовидная структура позволяет быстро перейти к нужному месту, временно отключить или активировать выполнение отдельных участков скрипта.

Язык UniTesS script – это простой скриптовый язык, с помощью которого описывается методика поверки, калибровки, проведения испытаний приборов. Файл, написанный на языке UniTesS script, исполняется только в ПО UniTesS АРМ.

Процесс создания **скрипта** состоит из нескольких этапов:

1. описание функций, непосредственно реализующих методы измерений;
2. определение контрольных точек и допусков;
3. создание специальных комментариев в скрипте для удобного отображения процесса испытаний в режиме “**Simple View**”.

Перед запуском скрипта происходит процедура его инициализации. Очищаются значения всех ячеек памяти, сбрасываются все установленные флаги. Отображается окно ввода основных параметров текущего измерения для отчета (данные для шапки протокола, опционально), например:

- Влажность;
- Давление;
- Калибратор;
- Методика поверки;
- Температура.

3.1. Общие правила синтаксиса

Каждая команда языка записывается в отдельную строку и выполняется последовательно, построчно. В любой строке скрипта может находиться только одна команда. Исключение возможно только для математических операторов, они разделяются в строке символом точки с запятой.

В качестве разделителя параметров и атрибутов в командной строке служит символ пробела или табуляции. При описании переменных, массивов данных - запятые и точки с запятой. В некоторых случаях, например, при описании математических операций, разделители могут опускаться. Это допускается правилами языка, но для наглядности и удобства восприятия текста рекомендуется их использовать.

Числа в скрипте можно вводить в трех форматах:

- обыкновенное число с целой и дробной частью, отделенной запятой либо точкой;
- в виде числа с постфиксом;
- в экспоненциальной форме.

Постфиксы можно использовать как русские, так и английские:

Рус.	Англ.	Описание	Степень	Пример числа
п	p	пико	10^{-12}	0,000000000001
н	n	нано	10^{-9}	0,000000001
мк	u	микро	10^{-6}	0,000001
м	m	милли	10^{-3}	0,001
к	k	кило	10^{3}	1000
М	M	Мега	10^{6}	1000000
Г	G	Гига	10^{9}	1000000000

Разделителем дробной части числа может служить точка либо запятая.

Например:

```
12 500 000 = 12,5M;
0,00005 = 50m;
3,205;
```

```
100;
5.5;
13e-3.
```

Данные могут быть представлены в виде строк, чисел, а также массивов.

Для удобства разработки скриптов предлагается свободно распространяемый текстовый редактор Notepad++, с поддержкой правил особой подсветки синтаксиса для скриптов UniTesS. При разработке скриптов в Notepad++ программист получает большую наглядность готового кода, в котором проще и быстрее ориентироваться. Редактор Notepad++ автоматически выделяет текст комментариев зеленым цветом, названия ячеек памяти - синим, а все команды скрипта и математические операторы - черным. Команды дополнительно выделяются жирным шрифтом.

ПО UniTesS накладывает определенные ограничения на работу с числовыми данными. Так, средства ПО позволяют работать только с 16 знаками числа. Аналогичные ограничения на количество значащих цифр накладываются при получении данных из приборов.

Пример представления числа:
123.45678901234568E-3 – 17 знаков мантиссы числа и показатель степени.

Рекомендуется располагать все команды настроек и конфигурации в первых строках, в самом начале скрипта. Если такие команды отсутствуют, то будут использованы значения по умолчанию.
Для игнорирования всех последующих символов в строке ставится знак комментария “**#**”. Регистр символов при написании команд и аргументов не критичен. Например, с точки зрения синтаксиса все варианты написания команды Math, math, MATH, MatH - корректны.

Основу скрипта UniTesS составляют функции, непосредственно реализующие методы измерения по конкретным параметрам, которые включают определенный набор команд и логических операторов в соответствии с алгоритмами и методиками поверки. Для вызова функции используется ключевое слово (команда) **Call**. Обычно в функцию необходимо передать некоторые параметры. Для каждой функции определяется свой собственный набор и порядок следования параметров. На выходе функция обычно формирует строку для занесения данных в отчет.

Перед загрузкой скрипта текстовый препроцессор выполняет проверку синтаксиса. В случае обнаружения ошибки выдается соответствующее сообщение и загрузка не происходит. Сообщение об ошибке содержит подробное описание и номер строки скрипта, в которой она обнаружена.

Текст скрипта можно условно разделить на несколько составляющих:
- команды,

- комментарии,
- метки.

Блок исполняемых команд скрипта обязательно должен заканчиваться специальной меткой **EndScript**. После этой метки могут располагаться описания функций, настройки прерываний и поведения функций. В теле скрипта **EndScript** может использоваться несколько раз. При переходе на строку с меткой **EndScript**, где бы она ни находилась, выполнение скрипта прекращается.

Скрипты имеют строгую архитектуру, которую можно сформулировать в виде требований:

- Измерение каждого параметра выполняется функцией, вызываемой из скрипта с определенным набором аргументов.
- Каждая функция реализует методику измерения конкретного параметра.
- Каждая функция, реализующая измерение параметра, должна:
  - быть подписана символом **#\F** для отображения в дереве;
  - иметь команду **Report** для отправки данных в **протокол**;
  - если **протокол** содержит заключение о соответствии, необходимо использовать команду **Compare**.

Метки служат для удобства организации программных переходов к определенной строке скрипта посредством специальных команд **GoTo** и **GoToOne**. Метка может состоять из букв латинского алфавита и цифр. Каждая метка обязательно начинается с символа двоеточия (“**:**”).

Пример:

```
:Metka1
:Metka23
:ABC
```

В некоторых случаях (для отладки скрипта, поиска и устранения ошибок, детального анализа эксперимента) возникает необходимость подробно записывать все команды и результаты вычислений АРМ. Для включения логирования используется команда:

```
ScriptLog Enable
```

Лог файл по умолчанию сохраняется в папку:

```
c:\UniTesS\Logs\Scripts
```

Имя файла формируется автоматически по правилам:

```
модель_серийный_номер_дата_время.txt
```

3.2. Работа с памятью

При написании скрипта программисту доступны ячейки памяти, которые служат местом хранения данных, результата математических операций, операции сравнения, а также ответа, полученного от приборов.

Данные в ячейках памяти могут быть представлены отдельными числовыми значениями, массивами чисел или строками. В режиме редактирования скрипта (окно вызывается нажатием клавиш <**Ctrl+E**>), пользователь может изменять значения ячеек памяти, а также просматривать данные в графической форме. Для этого на закладке “MEM” нужно выбрать соответствующее поле справа от наименования ячейки памяти (Рис. 3.1.).

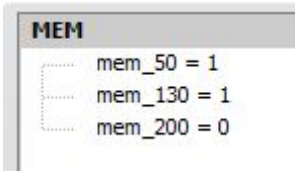


Рис. 3.1. Значения ячеек памяти.

Основные свойства ячеек памяти:

- отсутствует привязанный тип данных,
- нет необходимости определять заранее,
- каждая ячейка памяти является глобальной переменной.

Каждой ячейке памяти присваивается уникальное имя, которое начинается с “**mem_**” и содержит набор цифр. Например:

```
mem_1
mem_2015
mem_987
```

3.3. Режим редактирования скрипта

Режим редактирования доступен только разработчикам. Необходимость после каждого внесения изменений заново загружать скрипт в программу в процессе его отладки может вызывать серьезные неудобства. Во избежание этого в функционал UniTesS добавлена возможность редактировать скрипт в программной среде интерпретатора. Доступно как редактирование отдельных строк, так и удаление (со стандартными операциями копирования и вставки) целых фрагментов текста.

Переход в режим редактирования осуществляется нажатием комбинации клавиш <**Ctrl+E**>. При этом окно программы разделяется на две части. Справа появляется текст самого скрипта с кнопками инструментов редактирования.

В левом окне можно выбрать режим отображения (Рис. 3.2.):
Project - детальная информация по текущему проекту, каждый проект представляет собой отдельный скрипт,
MEM - содержимое ячеек памяти,
Define - набор системных переменных, обозначения, описания, параметры,
Reports - отчеты,
Breakpoints - точки прерываний.

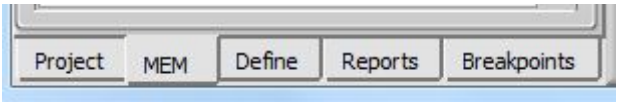


Рис. 3.2. Выбор режимов отображения служебной информации.

Внешний вид окна редактирования скрипта приведен на Рис. 3.3.

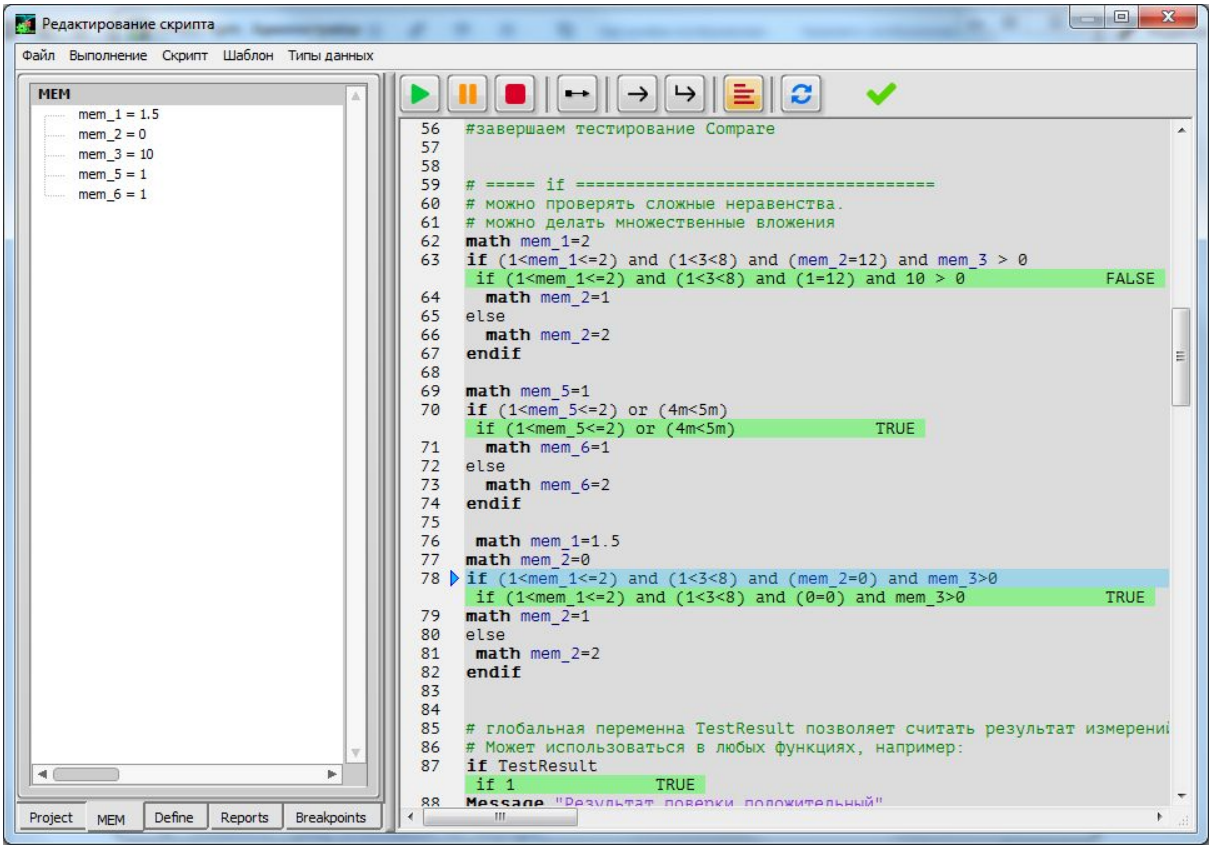


Рис. 3.3. Режим редактирования скрипта.

Инструменты редактирования:

- Запустить скрипт;
- Приостановить скрипт;
- Остановить скрипт и закрыть все сессии;
- Изменить строку исполнения на текущую;
- Выполнить текущую строку;
- Выполнить текущую строку и перевести курсор на следующую;
- Включить/Отключить аннотации;
- Обновить (перерисовать);
- Индикатор инициализации скрипта.

Внешний вид панели инструментов редактирования приведен на Рис. 3.4.

Рис. 3.4. Панель инструментов.

После редактирования скрипта его можно сохранить прямо из главного окна программы. Для этого выберите Меню "Скрипт" -> "Сохранить скрипт на диск". Стандартная комбинация клавиш <**Ctrl + S**> сохраняет скрипт в базе данных.

Раздел проектов (Рис. 3.5.) открывается закладкой “Projects” и содержит информацию о структуре скрипта, перечень всех функций с возможностью быстрого перехода к описанию каждой из них.

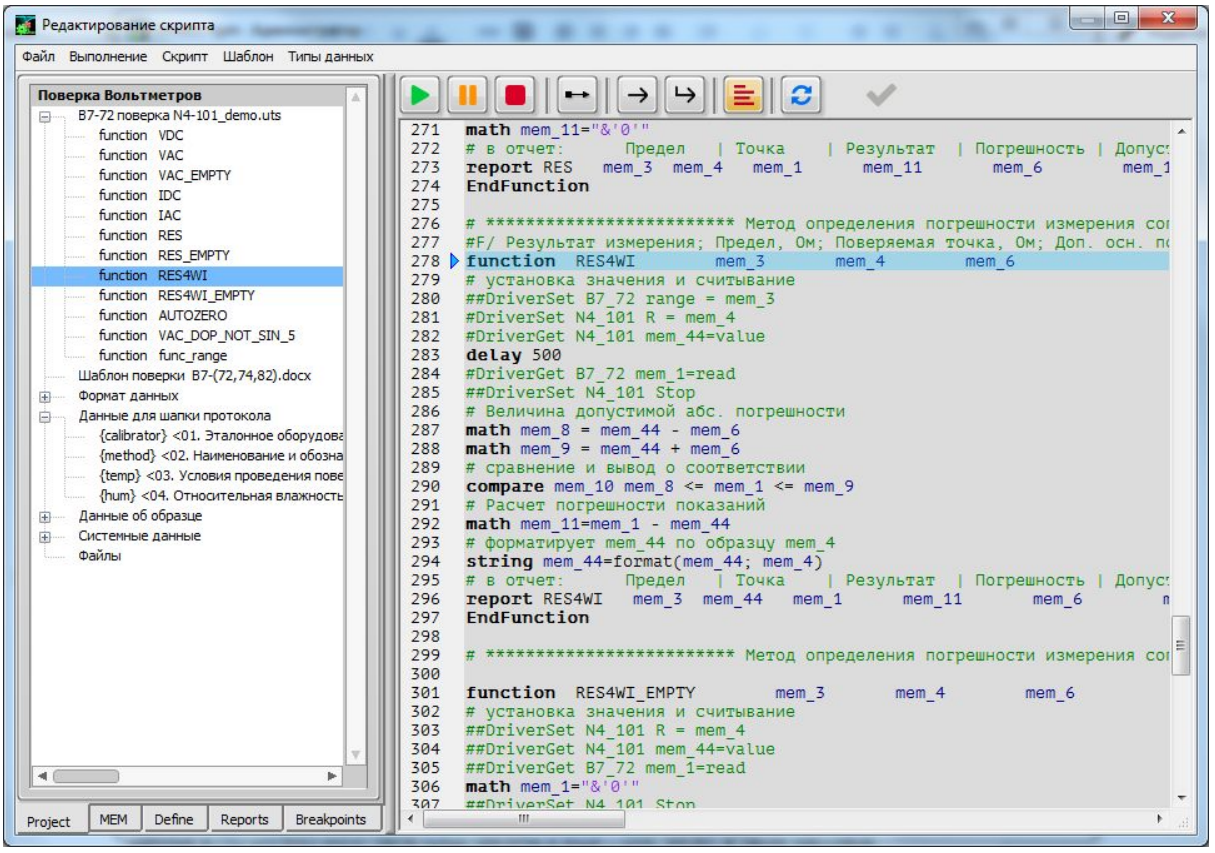


Рис. 3.5. Раздел проектов.

Также в этом разделе хранится набор данных для шапки протокола (если он есть), формат данных с описанием метода занесения в протокол, данные об образце и системные данные. Есть быстрый доступ к самому файлу протокола для его редактирования.

3.4. Специальные средства скрипта для отображения в древовидной структуре

Для удобного и наглядного представления алгоритма работы скрипта он отображается в виде дерева. В древовидной структуре отображаются только вызовы функций. Текстовый препроцессор ищет в скрипте специальные обозначения, которые связаны с функциями, чтобы они корректно отображались в дереве.

Описание функций

Для всех вызываемых функций, которые должны отображаться в дереве, необходимо добавить специальный комментарий, отмеченный в начале символами “**#F/**”. Этот комментарий ставится непосредственно перед строкой описания вызываемой функции. Далее в строке перечисляются поля, разделенные точкой с запятой, которые появятся в дереве как подписи колонок в группе вызовов функции: названия функции и передаваемых в нее параметров.

Синтаксис:

```
#F/ < Название функции >; <передаваемый параметр 1>; <передаваемый
 параметр 2>; … <передаваемый параметр N>.
<описание вызываемой функции>
```

Пример:

```
#F/ Измерение погрешности к-та отклонения; Амплитуда напряжения, В
 ; Частота сигнала, Гц ; Развертка осциллографа, 1/с
function DeflectionFactor   mem_2    mem_3       mem_4
…
EndFunction
```

3.5. Разметка структуры дерева

Для разметки структуры дерева используются специальные символы комментариев вида “**#/**”. Строка, которая начинается с такого символа, прописывается перед строкой вызова функции **Call**. Таких строк может быть несколько. Для перехода на нижний подуровень дерева используется символ “**–>**”, для выхода на более высокий уровень – символ “**<–**”. Символ перехода может отсутствовать, в этом случае название задается в текущем подуровне. Количество уровней перехода внутри дерева определяется количеством стрелочек “**<**”. Соответственно, допускается только одна стрелка перехода вправо “**>**” - смещение на один подуровень за один шаг (**–>**), а стрелок влево “**<**” может быть больше, для выхода на любой вышестоящий уровень (**<<–**). При форматировании древовидной структуры в разработке скрипта необходимо отслеживать текущий уровень дерева, который соответствует каждому участку скрипта, чтобы согласовать правильное количество и направление символов перехода. После обозначения нового уровня дерева рекомендуется ввести название подраздела для отображения в окне **Simple View**. Название может отсутствовать, в этом случае подраздел останется безымянным.

Пример сдвига по дереву:

```
#/-->    --> сдвиг по дереву вправо
#/<--    <-- сдвиг по дереву влево, возврат на первый уровень
#/-->    --> сдвиг по дереву вправо
#/-->    --> сдвиг по дереву вправо
#/-->    --> сдвиг по дереву вправо
#/<<<--  <<<-- три сдвига по дереву влево, возврат на первый
уровень
```

Пример 1:

```
#/ 1 канал
#/ -> Входное сопротивление 1 МОм
call DeflectionFactor    0,452         2m        4
call DeflectionFactor    0,012         2m        4
#/ Входное сопротивление 50 Ом
call DeflectionFactor    0,012         2m        4
call DeflectionFactor    0,452         5m        4
#/ <<- 2 канал
#/ -> Входное сопротивление 1 МОм
call DeflectionFactor    0,012       2m          4
call DeflectionFactor    0,452       5m          4
```

Пример 2:

```
# * Определение погрешности измерения напряжения *
#/ -> Поверка вольтметра В7-72 по интерфейсу. Эталон: Fluke 5720A
#/ -> ПОВЕРКА ПО НАПРЯЖЕНИЮ
#/ -> Напряжение: постоянное
#/ -> Предел измерения: 200 мВ
…
#F/ Установить значение ; предела:;
function func_range mem_1
```

Результат формирования дерева из Примера 2 можно увидеть на рисунке 3.6. Чтобы, при необходимости, распределить текст строки комментария по колонкам, используется символ точки с запятой (**;**).

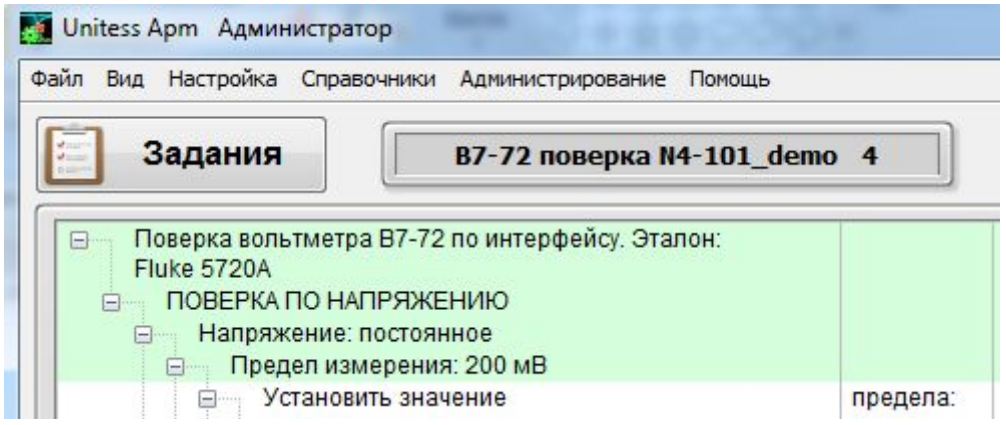


Рис. 3.6. Формирование дерева.

Управляя древовидной структурой (Рис. 3.7.) при помощи специальных значков (**“+”** и **“-”**), расположенных в узловых точках, можно “свернуть” ее к компактному виду, либо привести к развернутому.

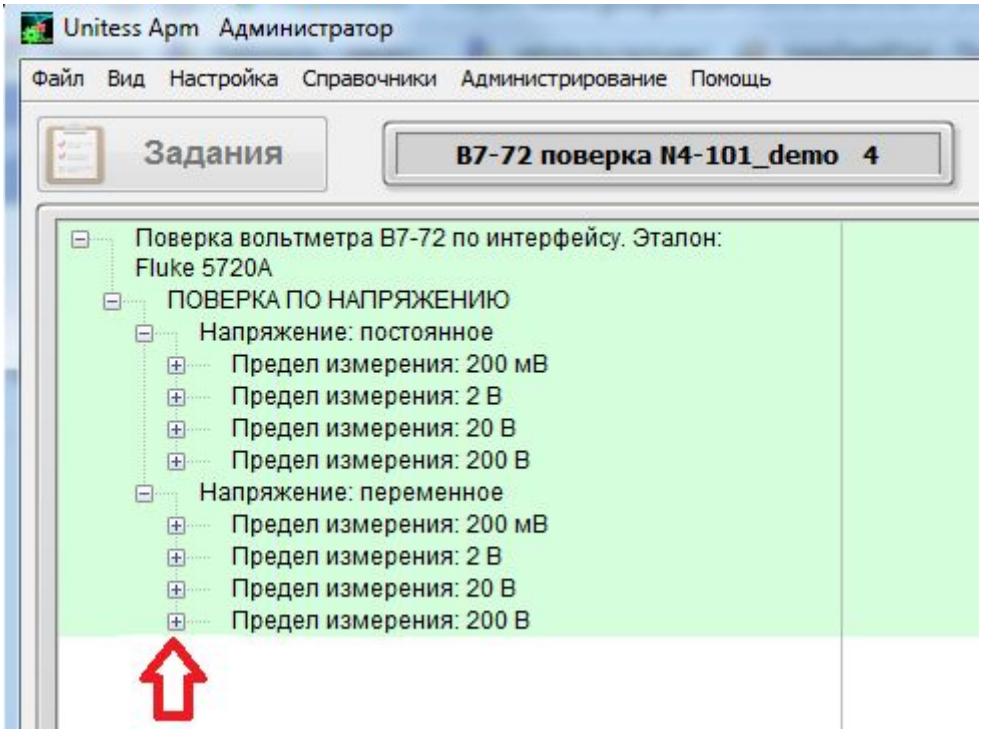


Рис. 3.7. Управление отображением древовидной структуры.

Существует взаимосвязь между древовидной структурой и текстом скрипта. Помимо наглядности, возможности на каждом шаге контролировать проводимый эксперимент и сложную последовательность измерений, дерево позволяет быстро перейти к нужному месту (конкретной строке) в тексте скрипта для его отладки или внесения корректировок.

4. Описание команд

4.1. Define - определение

Очень часто возникает необходимость в начале скрипта описать, определить значения некоторых констант (цифровые или текстовые параметры) для его первичной настройки. Эти значения впоследствии будут использоваться в вычислениях, но не изменяться. Например, коэффициенты калибровки приборов, их названия и т.д. Для этого при помощи команды **Define** указывается наименование и значение через пробел.

Синтаксис:

```
Define <наименование> <значение>
```

<наименование> - присваиваемое определению имя,
<значение> - текстовая строка, число или массив, связанный с этим именем в скрипте.

В таком виде указанное значение и будет подставлено текстовым препроцессором перед каждым выполнением команды, в которой встречается его название. Если значение заключено в кавычки, так оно и будет подставлено в команду. Никаких синтаксических проверок не производится.

Через **Define** можно определять даже участки программного кода. Перед выполнением команды текстовый препроцессор UniTesS APM ищет строку с **Define** и просто заменяет значение.

Это может быть полезно, например, при выборе конкретной модели драйвера прибора перед началом измерений:

```
define calibrator fluke5000
driverset calibrator VDC=10
```

Теперь, чтобы перенастроить весь скрипт на работу с другим оборудованием (к примеру, сменить модель калибратора), пользователю необходимо в начале текста скрипта изменить только одну строку с **Define**:

```
define calibrator B1_28
```

Примеры использования:

```
define offset  1,4м
```

```
define param_name  "Вертикальное отклонение"
define param_value  12
define Channel_Name  Канал 1
define array [1,1;2.2;4.3;5.5;6.5;7]
define CalibrEn  true
```

Важно! Следует быть внимательным при выборе имени **Define**. Нельзя использовать имена функций, математических операторов, драйверов и т.д. Ранее определенные **Define** можно использовать в любых командах и функциях. Повторное определение **Define** в скрипте (переопределение) не допускается, это может привести к ошибке.

Пример:

```
if CalibrEn
message Откалибруйте Channel_Name
endif

Call TestFunc             offset      array

Function TestFunc         mem_1         mem_2
Math mem_3 = size(mem_2)
Math mem_5 = mem_1 * 2
Math mem_7 = mem_3 + mem_5
EndFunction
```

4.2. Function - описание пользовательской функции

Команда предназначена для определения функции в скрипте. Пользователь присваивает имя функции, указывает количество входных параметров и ячейки памяти, в которые они будут записаны при входе в функцию, а также последовательность команд, которые выполняет данная функция.

Скрипт UniTesS имеет строгую архитектуру. Измерения каждого параметра должны выполняться отдельной функцией, вызываемой с определенным набором аргументов. Каждая функция реализуют методику измерения конкретного параметра. Это накладывает ряд требований. Каждая функция, реализующая измерения параметра, должна:

- быть подписана комментарием, который начинается с символа **#\F**, для отображения в дереве,
- иметь команду **Report** для отправки данных в протокол,
- если протокол содержит заключение о соответствии, то необходимо использовать команду **Compare**.

Так как функция описывает конкретный метод измерения, рекомендуется присваивать имена, связанные с выполняемой задачей. Например:
VDC - измерение постоянного напряжения,
VAC - измерение переменного напряжения,
IDC - измерение постоянного тока,
IAC - измерение переменного тока.

Синтаксис:

```
Function <имя функции> <переменная 1> <переменная 2> … <переменная n>
<тело функции>
EndFunction
```

Описание полей:

<имя функции> – имя функции, назначаемое пользователем;
<переменная n> – ячейка памяти для хранения входного параметра;
<тело функции> – может состоять из множества строк и содержать любую последовательность команд. Вложенность вызовов функций для языка UniTesS допускается.

Пример:

```
# * Определение времени нарастания переходной характеристики осциллографа *
# Значение    | К-т отклонения | Допустимое время.
# напряжения,В| В/дел.         | нарастания,с
#F/ Время нарастания; Напряжение, В; К-т отклонения, В/дел.; Допустимое знач. времени
```

```
нарастания, с
Function StepResponseTime    mem_2        mem_3        mem_4
IVIdriverSet scope VerticalRange= mem_3;
math mem_5 = mem_2 / 2
IVIdriverSet scope triggerlevel=-mem_5
DriverSet 9100 ScopeEdge= 100n mem_2 pos
IVIdriverSet scope  runctr=run; acqinit
Delay 3000
IVIdriverGet scope mem_1 = RISETIME
DriverSet 9100 Stop
compare mem_10 mem_1 <= mem_4
# в отчет: № Канала | К-т отклонения | Значение | Полярность | Допустимое время |
Измеренное  | вывод
# |  В/дел.   | напряжения,В |фронта    | нарастания,нс    | значение,нс |
report StepResponseTime mem_9 mem_3  mem_2 "полож." mem_4 mem_1  mem_10
EndFunction
```

4.3. Call - вызов функции

Команда предназначена для вызова функции, ранее определенной пользователем.

Вызываемой функции необходимо передать параметры, определенные пользователем в заголовке функции.

Синтаксис:

```
Call <имя функции> <параметр 1> <параметр 2> … <параметр n>
```

Описание полей:

<имя функции> – имя функции, определенной пользователем;
<параметр n> – параметр, передаваемый в функцию. Количество входных параметров определяется пользователем в заголовке функции.

Пример 1:

```
#                    Значение      | К-т отклонения | Допустимое
время
#                  напряжения,В   | В/дел.         |   нарастания,с
call StepResponseTime     150m                20m              1,2n
call StepResponseTime     300m                50m              1,2n
call StepResponseTime     600m               100m              1,2n
```

Пример 2:

```
# Погрешности калибратора на диапазоне от 0 до 202 мВ
#   Относительная погрешность калибратора
math mem_41 = 30мк
#   Аддитивная погрешность калибратора
math mem_42 = 3,6мк
math mem_300 = 0
# Определение погрешности измерения напряжения постоянного тока.
#  Предел измерения, В | Калибруемая точка | Дискретность счета
call VDC_cal  200м     20,0м           0,1м
call VDC_cal  200м    100,0м           0,1м
call VDC_cal  200м    190,0м           0,1м
call VDC_cal  200м   -190,0м           0,1м
```

4.4. Interrupt - прерывания

Для оптимизации алгоритма выполнения скрипта и значительного уменьшения программного кода существует возможность настройки особого поведения функций. Часто возникают ситуации, когда при первом вызове определенной функции необходимо выполнить какие-либо действия, например подключить, подготовить прибор или ввести данные, но все последующие вызовы происходят без таких настроек или действий. В этом случае поможет описание прерываний при помощи **Interrupt**. Для каждой функции можно настроить несколько прерываний.

Синтаксис

```
Interrupt <параметр 1> <имя функции> <параметр 2>
<действие>
EndInterrupt
```

Описание полей:

<имя функции> – имя функции, определенной пользователем,
<параметр 1> - может принимать значения **before** (наступает перед выполнением функции) или **after** (после выполнения функции),
<параметр 2> - описывает особое **условие** прерывания,
<действие> - набор выполняемых операторов.

Условия прерывания функции (параметр 2):

first_in_script - только при первом вызове функции в скрипте,
last_in_script - только при последнем вызове функции в скрипте,
first_in_block - прерывание будет выполнено, если функция вызывается первый раз в блоке. Блок в данном случае - последовательность вызовов функций,
last_in_block - прерывание будет выполнено, если функция вызывается последней в блоке,
argument_change_n - прерывание будет выполнено, если был изменен аргумент функции с индексом n, (n = 1, 2, 3...),
argument_sign_change_n - прерывание будет выполнено, если изменился знак аргумента функции с индексом n.

Все условия могут комбинироваться через логические операторы **AND**, **OR** или **NOT**. Также допускается использование значения ячеек памяти **mem_n**. Описание прерывания обязательно должно заканчиваться **EndInterrupt**.

Условие argument_sign_change_n обычно применяется, если необходимо отследить изменения полярности сигнала и попросить пользователя перекоммутировать схему подключения. Флаги argument_sign_change_n и argument_change_n могут

выставляться, в том числе, и при условии начала блока (first_in_block=1). Если установлен флаг first_in_script, то все остальные флаги снимаются. Если функция уже имеет флаг last_in_script, то last_in_block не устанавливается.

Пример:

```
Interrupt <before> VDC <first_in_script>
Message “Включите питание прибора”
EndInterrupt

Interrupt before DeflectionFactor  first_in_script
Message прерывание при первом вызове функции в скрипте
EndInterrupt

Interrupt before DeflectionFactor  first_in_block
Message прерывание при первом вызове функции в блоке
EndInterrupt

Interrupt before DeflectionFactor (mem_3>0.001) and (mem_3<0.005)
Message прерывание если mem_3 в диапазоне 0.001 - 0.005
EndInterrupt

Interrupt before DeflectionFactor argument_change_1 and (not
argument_sign_change_1)
Message прерывание при изменении аргумента номер 1, но без
изменения знака
EndInterrupt

Interrupt before DeflectionFactor  argument_sign_change_1
Message прерывание при изменении знака аргумента номер 1
EndInterrupt

Interrupt after DeflectionFactor last_in_script
Message прерывание если функция была выполнена последний раз за
скрипт
EndInterrupt

Interrupt after DeflectionFactor last_in_block
Message прерывание если функция была выполнена последний раз за
блок
EndInterrupt
```

4.5. Delay - задержка

Команда предназначена для временной остановки, паузы в ходе выполнения скрипта. Это может понадобиться, например, при ожидании ответа от некоторых приборов.

Синтаксис:

```
Delay <время задержки>
```

Описание полей:

<время задержки> – требуемое время задержки в миллисекундах.

Пример:

```
Delay 3000       # Задержка 3 секунды
Math mem_1 = 2000
Delay mem_1      # Пауза 2 секунды
```

4.6. Math - вычисление математических операций

Команда предназначена для выполнения математических операций над аргументами, хранимыми в ячейках памяти **mem_n**.

Синтаксис:

```
Math <ячейка памяти> = <формула>
```

Описание полей:

<ячейка памяти> – имя одной из внутренних ячеек памяти ПО, в которую будет помещен ответ,
<формула> – математическое выражение, которое может содержать числа, переменные в виде ячеек памяти ПО, знаки математических операций, обозначения математических функций, а также константу, которая ранее была определена при помощи **Define**.

Знаки математических операций:

Символ	Действие	Пример
+	сложение	math mem_1 = 5 + 6
-	вычитание	math mem_1 = 8 - 6
*	умножение	math mem_1 = 2 * 2
/	деление	math mem_1 = 50 / 5
^	возведение в степень	math mem_1 = 2 ^ 2

Если при возведении в степень показателем является отрицательное дробное число, желательно заключить его в скобки.

Перечень математических функций приведен в таблице (Приложение №1).
Правила и форматы записи математических операций для команды **Math** приведены в таблице (Приложение №3).

Пример:

```
Math mem_1 = 0.5
Math mem_2 = mem_1 / 3
Math mem_3 = cos(mem_1) + mem_1 * 2 + 1
```

```
Math mem_1 = mem_2 + mem_3
Math mem_4 = 12M
Math mem_5 = 82
Math mem_6 = mem_5 / 4.5
Math mem_7 = ceil((abs(mem_3)*(mem_4/100)+mem_5*mem_6)/mem_6)
Math mem_2 = 12,5m
Math mem_11 = mem_2 * 2
Math mem_12 = [23; 458; 12.6; 400; 15]
math mem_33 = mem_12[3] + mem_12[2]
math mem_2=5; mem_1 = mem_2^2
```

Для тригонометрических функций значение аргумента принимается в радианах.

Кроме математических также доступны дополнительные функции работы с массивами. Перечень возможных операций и описание функций для работы с массивами приведены в таблицах (Приложение №2).

Все аргументы - числа, названия ячеек памяти и знаки операций в формулах могут отделяться друг от друга пробелами, знаками табуляции или вводиться слитно.

Пример вычисления величины допустимой абсолютной погрешности:

```
math mem_8 = mem_4 - mem_6
math mem_9 = mem_4 + mem_6
```

где:
mem_4 - поверяемая точка,
mem_6 - допустимая погрешность,
mem_8 - минимальное допустимое значение,
mem_9 - максимальное допустимое значение,

Ячейка памяти может хранить текст или численное значение. Поэтому следует соблюдать осторожность и учитывать это обстоятельство при описании математических операций.

```
math mem_1 = "Вольтметр"
math mem_2 = 100
math mem_3 = mem_1 + mem_2
          mem_3 = "Вольтметр" + 100;
math mem_3 = mem_3 * mem_3
          mem_3 = "Вольтметр + 100" * "Вольтметр + 100";
```

Если при вычислениях среди аргументов появится хотя бы один элемент в текстовом формате, ячейка памяти с конечным результатом будет содержать только текст.

Допускается указывать несколько операций внутри одного оператора **Math**. В качестве разделителя используется точка с запятой. Например:

```
math mem_10=1; mem_11=2; mem_12=3
```

Очистка ячейки памяти:

```
math mem_1=""
math mem_1=nan
```

Можно выполнять групповые математические операции над массивами.

```
define array [1;2;3;4;5]

math mem_100=array
math mem_200=[10;20;30;40;50]
math mem_300=mem_100+mem_200
          mem_300 = [1; 2; 3; 4; 5]+[10; 20; 30; 40; 50]          mem_300 = [11; 22; 33; 44; 55];
math mem_300=mem_300+1
          mem_300 = [11; 22; 33; 44; 55]+1          mem_300 = [12; 23; 34; 45; 56];
math mem_300=mem_300*2
          mem_300 = [12; 23; 34; 45; 56]*2          mem_300 = [24; 46; 68; 90; 112];
```

Результатом сложения, вычитания, умножения либо деления двух массивов будет массив, каждый элемент которого представляет собой сумму, разность, произведение либо частное операции, произведенной поочередно над соответствующими элементами исходных массивов. В случае, если число элементов одного из исходных массивов окажется больше, все последующие элементы результирующего массива вычисляются как результат операции с последним элементом массива меньшей длины.

Например:

```
math mem_100=[1;2;3]
math mem_200=[10;20;30;40;50]
math mem_300=mem_100+mem_200
```

В результате массив в ячейке памяти mem_300 будет выглядеть так:

```
[11;22;33;43;53]
```

Некоторые математические операции могут приводить к неоднозначному результату, например при вычислении значения арксинуса аргумент должен находиться в пределах от -1 до +1. В случае, если в формулу попадет аргумент, значение которого выходит за эти границы, в результирующей ячейке сохраняется “NAN”. Результат деления на ноль сохранит “INF”. Служебные слова NAN и INF являются зарезервированными, их не рекомендуется использовать в качестве значений переменных. Это может привести к ошибке.

```
math mem_1=12
math mem_2=asin(mem_1)
          mem_2 = asin(12)          mem_2 = NAN;

math mem_3 = mem_1 / 0
          mem_3 = 12 / 0          mem_3 = INF;
```

Использование в математических расчетах ячеек памяти, которые после каких-либо операций уже содержат NAN или INF, также приведет к ошибке выполнения скрипта.
NAN = Not Acceptable Number (недопустимое число),
INF = INFinity (бесконечность).

4.7. Compare - сравнение

Команда **Compare** служит для выполнения операций сравнения над числами и ячейками памяти, например при проверке результата измерения на выход за допустимые границы. Результат сравнения заносится в ячейку памяти в текстовом формате. Варианты текстового обозначения:
"Соот." (соответствует)
или
"Не соот." (не соответствует)
импортируются из настроек отчета базы данных UniTesS DB для отдела текущего пользователя.
При выходе за допустимые границы выводится информационное окно подтверждения отрицательного результата и пользователь может повторить измерение, согласиться и продолжить или приостановить работу скрипта. После повторного вычисления результат будет занесен в отчет.

При повторном измерении UniTesS APM повторяет тело функции, в котором находится команда **Compare**. Рекомендуется использовать команду **Compare** только в теле функции. В случае, если команда **Compare** вызывалась не из функции, UniTesS APM повторяет участок кода до предыдущей команды **Compare** или **Report**.

Настоятельно рекомендуется ограничить количество вызовов команды **Compare** - не более одного раза за каждый цикл функции.

В выражении может присутствовать как одна операция сравнения, так и две (двойное неравенство). Обрабатываются они в порядке слева направо. При невыполнении хотя бы одного условия принимается решение об отрицательном результате.

Синтаксис:

```
Compare <ячейка памяти> <операнд 1> <символ операции> <операнд 2>
(<символ операции> <операнд 3>)
```

Описание полей:
<ячейка памяти> – имя одной из внутренних ячеек памяти (mem_1, mem_2, …), в которую будет помещен ответ,
<операнд n> – значения полей, необходимых для занесения в таблицу отчета. Поля могут быть ячейками памяти или представляться численными значениями. Значение, введенное в кавычках (например “abc”) будет определено как текстовая строка и без изменений добавлено в отчет,
<символ операции> – оператор сравнения [<=; >=; !=; <; =; >].

Поля разделяются пробелами или символом табуляции.

Допускается использование следующих операторов сравнения:
< - меньше
> - больше
<= - меньше или равно
>= - больше или равно
= - равно
!= - не равно

С помощью логических операторов возможны конструкции более сложных, составных неравенств:
|| - логическое "ИЛИ"
or - логическое "ИЛИ"
&& - логическое "И"
and - логическое "И"
not - логическое отрицание “НЕ”
! - логическое отрицание “НЕ”

Пример 1:

```
Math mem_3=6
Compare mem_2 3<=mem_3<=9
```

Результат (Соотв.), т.к. 3<=6<=9 , будет помещен в ячейку памяти mem_2.

ВНИМАНИЕ! Не рекомендуется использовать команду **Compare** вне тела функции. Также не рекомендуется использовать **Compare** внутри функции, не подписанной хэштегом (**#F/**). Следует помнить, что при повторе тела функции переменные mem заново не инициализируются значениями из команды ее вызова **Call**. Поэтому важно не изменять значения переменных, которые инициализируются при входе в функцию, в теле функции.

Перезапись переменных допускается только в случае необходимости передавать или корректировать значения из одной итерации в другую. Например, в процессе отладки скрипта, если параметр выходит за допустимые границы при текущих настройках, то при повторе функции можно попробовать их изменить.
Если команда **Compare** находится в теле функции, то при выводе предупреждения о несоответствии пользователю сообщается и название функции.

Формат сообщения:
диап. точка погр.
(диапазон - точка - погрешность)

В составных неравенствах обязательно использование скобок для выделения групп:

```
Compare mem_1 (1<=2) and (2<=4)
```

Применяются следующие флаги:
True - истина,
False - ложь,
TestResult - глобальный флаг результата тестирования.

Пример 2:

```
compare mem_1  (1<3<5) or (1<4)
math mem_2=10
math mem_3=11
compare mem_1 mem_2<mem_3
```

Существует возможность задавать формат вывода чисел на экран (при отрицательном результате):
example - пример числа,
format - формат.

Внимание! Форматируются только значения переменных mem. Оба ключа - опциональные. Сравнение происходит с учетом формата.

```
compare mem_1  mem_2>mem_3 example=1.00м
compare mem_1  mem_2>mem_3 example=1.0
compare mem_1  mem_2>mem_3 format=%,;%.3r
compare mem_1  mem_2>mem_3 example=1.000 format=%,;%.2d
```

```
compare mem_1  mem_2>mem_3 example=1.00м
          12000,00м>10000,00м          mem_1=Соотв.
compare mem_1  mem_2>mem_3 example=1.0
          12,0>10,0          mem_1=Соотв.
compare mem_1  mem_2>mem_3 format=%,;%.3r
          12,000>10,000          mem_1=Соотв.
compare mem_1  mem_2>mem_3 example=1.000 format=%,;%.2d
          12,00000>10,00000          mem_1=Соотв.
```

Отрицательный результат с выводом подтверждения (Рис. 4.1.):

```
math mem_3 = 6
math mem_2 = 4
compare mem_1 mem_3<mem_2
```

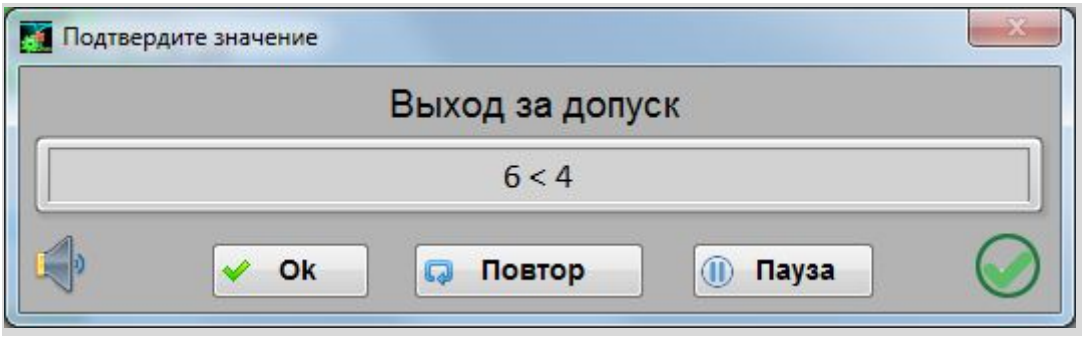


Рис. 4.1. Окно запроса подтверждения результата.

Отрицательный результат с выводом подтверждения, но без повтора точки измерений:

```
compare mem_1 mem_3<mem_2 NoRepeat
```

Отрицательный результат без вывода окна подтверждения негативного измерения:

```
compare mem_1 (mem_3<mem_2) and (3<1)  NoRequest
```

Глобальная переменная **TestResult** хранит результат измерений.
1 - положительный,
0 - отрицательный.
Она может использоваться в любых функциях, например:

```
math mem_2=7
math mem_3=6
compare mem_1 mem_3<mem_2
math mem_1 = TestResult
```

Можно вводить описание неравенства (Рис. 4.2.) для конечного пользователя, но описание только для частей, разделенных логическим "И" (AND).

Пример сложного неравенства:

```
compare mem_1 ((1<3<5) or (3<45)) and (6<4) description="по
напряжению"; "по току"
```

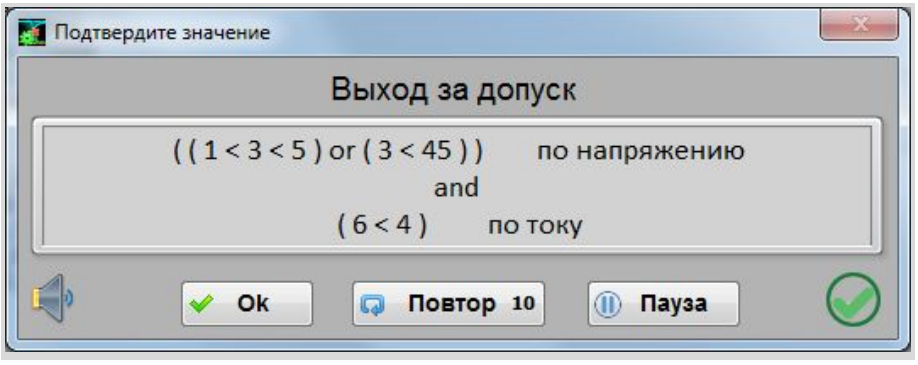


Рис. 4.2. Использование неравенства.

Пример функции сравнения и вывода о соответствии:

```
#F/ Пост напр; Предел, В; Повер точка, В; погр, В
function  VDC              mem_3       mem_4         mem_6
# Величина допустимой абс. погрешности
math mem_8 = mem_4 - mem_6
math mem_9 = mem_4 + mem_6
# сравнение и вывод о соответствии
compare mem_10 mem_8 <= mem_1 <= mem_9
EndFunction
```

где:
mem_1 - результат измерения прибора,
mem_4 - поверяемая точка,
mem_6 - погрешность измерения,
mem_10 - результат и вывод о соответствии.

4.8. Repeat - команда повторения

Команда предназначена для организации простых циклов. Если необходимо повторить набор некоторых действий несколько раз, для оптимизации кода удобно использовать данную команду.

Синтаксис:

```
repeat <количество повторов>
      <код цикла>
endrepeat
```

Описание полей:

<количество повторов> – целое число, количество повторений цикла,
<код цикла> – как видно из примера, в цикле могут находиться любые команды скрипта,
<endrepeat> – конец цикла.

Для принудительной остановки цикла на любом этапе его прохождения используется **StopRepeat**.

Пример 1:

```
math mem_2=12
repeat 5
math mem_1=1
math mem_3=mem_1+mem_2
math mem_2=mem_3
endrepeat
```

Пример 2:

```
math mem_1 = 200
math mem_2 = 0,1
math mem_3 = 0,1
math mem_4 = 2
   repeat 50
   call TestFunc  mem_1  mem_2  mem_3  mem_4
   math mem_1 = mem_1 + 1
   endrepeat
Function TestFunc  mem_1  mem_2  mem_3  mem_4
math mem_5 =   mem_1 + mem_2 + mem_3
math mem_6 = mem_5 + mem_4
```

```
EndFunction
```

Пример 3:

```
math mem_14=5 # количество повторов
  Repeat mem_14 # повторим mem_14 раз
  math mem_14=mem_14-1
      if mem_14<3
    StopRepeat
     endif
message "Всего mem_14 повторов. Если это число окажется меньше 3,
повтор остановится"
  EndRepeat
```

Пример 4:

```
math mem_1=1
math mem_2=2
math mem_10=0.1
math mem_11=0.2
repeat 10
math ++mem_10=mem_1
math mem_1=mem_1+mem_10[1]
math ++mem_11=mem_1-mem_2
math mem_2=mem_2+mem_11[1]
endrepeat
```

Пример 5:

```
math mem_10=""
math mem_11=""
math mem_27=1m
math mem_2=10
Message "Установите на магазине емкости значение mem_2."
repeat mem_2
Message mem_1 "Введите показания прибора"
math mem_1 = mem_1 * mem_27
math ++mem_10=mem_1
math mem_11=mem_10-mem_2
endrepeat
```

4.9. If, Case, CaseOne, GoTo - команды условного и безусловного перехода

В этих командах допускается использование следующих операторов сравнения:
< - меньше,
> - больше,
= - равно,
<= - меньше или равно,
>= - больше или равно,
!= - не равно.

Логические операторы позволяют создавать более сложные условия выполнения, конструировать составные неравенства:
not - логическое отрицание НЕ,
! - логическое отрицание НЕ,
|| - логическое ИЛИ,
or - логическое ИЛИ,
&& - логическое И,
and - логическое И.

В составных неравенствах обязательно использование скобок для выделения групп:

```
   (1<=2) and (2<=4)
```

Применяются следующие флаги:
true - истина,
false - ложь,
TestResult - глобальный флаг результата тестирования (принимает значение **0** при отрицательном или **1** при положительном результате).

4.10. Case - ветвление от условия

Операторы **Case** и **CaseOne** имеют одинаковый синтаксис, однако **Case** будет исполнять код во всех секциях, в которых неравенство принимает значение True, в том числе секцию default, а **CaseOne** выберет только одну.

Синтаксис.

```
case
when <условие1>
<действие1>
when <условие2>
<действие2>
when <условие3>
<действие3>
...
default
<действие по умолчанию>
endcase

caseone
when <условие1>
<действие1>
when <условие2>
<действие2>
when <условие3>
<действие3>
...
default  <действие по умолчанию>
endcase
```

Каждая секция действий, следующих за условием, обязательно должна начинаться с новой строки. Будут выполняться все строки до очередной строки с условием либо окончания команды.

Пример:

```
    math mem_11 = 9 # 1-ый аргумент
    math mem_12 = 8 # 2-ый аргумент
# сложное сравнение mem_8 и mem_9
    math mem_13 = mem_11 - mem_12 # разность mem_11 и mem_12
CASE
        when mem_13 > 0
```

```
            message "Большее число mem_11"
        when mem_13 < 0
            message "Большее число mem_12"
        default
            message "Числа равны mem_12"
endcase
```

4.11. GoTo - команда безусловного перехода

Команда безусловного перехода нарушает порядок выполнения команд в скрипте. Следующей выполняется команда, которая находится в строке после идентификатора <метка>.

Синтаксис.

```
goto <метка>
```

Внимание! Следует соблюдать осторожность при использовании команды безусловного перехода **GoTo**. Ее присутствие может приводить к неявным ошибкам в скрипте, например - нельзя использовать **GoTo** для выхода из функций.

Пример:

```
message mem_3 select=(1:5)  "Укажите номер метки"
Case
   when mem_3=1       math mem_1="metka1"
   when mem_3=2       math mem_1="metka2"
   when mem_3=3       math mem_1="metka3"
   when 1<=mem_3<=5
      CaseOne
      when mem_3=1            math mem_1="metka1"
      when mem_3=2            math mem_1="metka2"
      when mem_3=3            math mem_1="metka3"
      when mem_3=4            math mem_1="metka4"
      default
         math mem_1="metka5"
      EndCase
EndCase
goto mem_1
:metka1
math mem_1=1
goto metkaFin
:metka2
math mem_1=2
goto metkaFin
:metka3
math mem_1=3
goto metkaFin
:metka4
math mem_1=4
goto metkaFin
```

```
:metka5
math mem_1=5
:metkaFin
```

4.12. IF - ветвление от условия

Команда **If** позволяет проверять сложные неравенства.

Сравнение переменных, содержащих текстовые и числовые значения недопустимо, это может привести к возникновению ошибки при выполнении скрипта.

Синтаксис.

```
if <условие 1> <оператор> <условие 2> <оператор> <условие 3>...
<действие>
else <действие>
endif
```

<оператор> - логический оператор **AND**, **OR**,
<условие n> - проверка сравнения либо равенства ячеек памяти, числовых и текстовых значений.

Допускаются множественные вложения условий, в этом случае обязательно использование скобок.

Пример 1:

```
if (1<mem_1<=2) and (1<3<8) and (mem_2=0) and mem_3<0
  math mem_2=100
else
  math mem_2=200
endif
```

Пример 2:

```
math mem_5=1
if (1<mem_5<=2) or (mem_5>0)
  math mem_6=1
else
  math mem_6=2
endif
```

При сравнении переменных, имеющих текстовый формат, размер символов не учитывается.

Пример 3:

```
math mem_1="ТЕКСТ"
```

```
if mem_1="текст"
message "размер не имеет значения"
else
message "РАЗМЕР ИМЕЕТ ЗНАЧЕНИЕ"
endif
```

Внимание! Возможно только сравнение строк, использование операторов больше (**>**), меньше (**<**), больше или равно (**=>**), меньше или равно (**<=**) - недопустимо. Это может привести к ошибке при выполнении скрипта.

Пример определения наибольшего числа:

```
   Message mem_8 “Введите первое число”
   Message mem_9 “Введите второе число”
if mem_9 > mem_8
        math mem_10 = mem_9
else
        math mem_10 = mem_8
endif
    Message  "Наибольшее число: mem_10"
```

Глобальная переменная **TestResult** содержит результат измерений (**1** - положительный, **0** - отрицательный). Она может использоваться как аргумент в любых функциях или вычислениях, например:

```
if TestResult
 Message "Результат поверки положительный"
else
Message attention "Результат поверки отрицательный"
endif
Message  "Результат поверки = TestResult"
```

4.13. Message - вывод сообщений и запросов

Команда предназначена для вывода на экран информационных сообщений пользователю, либо запроса ввода информации, подтверждения действий пользователя. Дойдя до строки с командой **Message**, программа приостановит выполнение скрипта и выведет окно с сообщением пользователю. Сообщение может содержать напоминание о том, что для следующего измерения необходимо изменить схему подключения приборов, либо уведомление о настройке определенного режима. Может содержать изображение, например, схему правильного подключения приборов для следующего этапа измерения. Также окно может запрашивать ввод информации с клавиатуры или предлагать выбор варианта из пользовательского меню.
Само сообщение пользователю следует вводить в кавычках. Текст внутри кавычек обрабатывается программой, имена ячеек памяти заменяются значениями самих ячеек. Если сообщение было введено без кавычек, программа обработает всю строку целиком и заменит все mem_n на их текущие значения из памяти.

Синтаксис:

```
Message <ячейка памяти> <параметр> <pic=xxx.xxx> "<текст сообщения>"
```

Описание полей:
<ячейка памяти> – опционально, имя одной из внутренних ячеек памяти ПО, в которую будет помещено введенное значение или результат выбора из пользовательского меню.
Возможные параметры:
<checksyntax> – опционально, включает проверку корректности введенного числового значения,
<wait=n> – опционально, окно сообщения автоматически закроется спустя n миллисекунд,
<attention> – опционально, текст сообщения выводится красным цветом,
<lastvalue> – с подстановкой ранее введенного значения,
<defvalue=n> – вывод значения по умолчанию или позиция по умолчанию в меню, если оно описано в команде,
<selectmenu=NUM> – включает отображения меню для выбора определенных параметров. NUM - количество позиций меню,
<select=(n1;n2)> – диапазон выбора вводимого значения между n1;n2,
<LoadMenu = "FileName.csv"> - загружает меню из файла FileName с расширением csv,
<LoadMenuID = "FileNameID.csv"> - загружает меню из файла FileNameID с расширением csv,
“<текст сообщения>” - текст выводимого сообщения, может заключаться в кавычки,

<pic=xxx.xxx> - опционально, выводит изображение. Допустимы три формата изображений: bmp, jpg, png.

Все описания полей пользовательского меню начинаются с комбинации символов \n1. \n2. и т.д.
Если в списке аргументов ячейка памяти идет первой, команда **Message** запрашивает ввод ее значения. Все остальные ячейки памяти в командной строке вычисляются и их текущее значение выводится в сообщении.
Если команда содержит опцию отображения меню, то первая ячейка памяти сохраняет номер выбранной пользователем позиции меню (целое число - 1, 2, 3...).

При использовании параметра select выбор варианта осуществляется вводом с цифровой клавиатуры (рис. 8.3.).

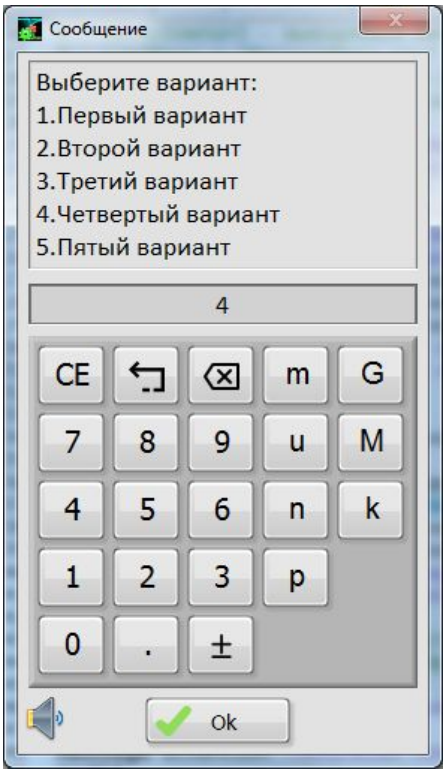


Рис. 4.3. Вывод меню командой Message.

Параметр selectmenu производит выбор из меню (Рис. 4.4.), описанного пользователем в команде.

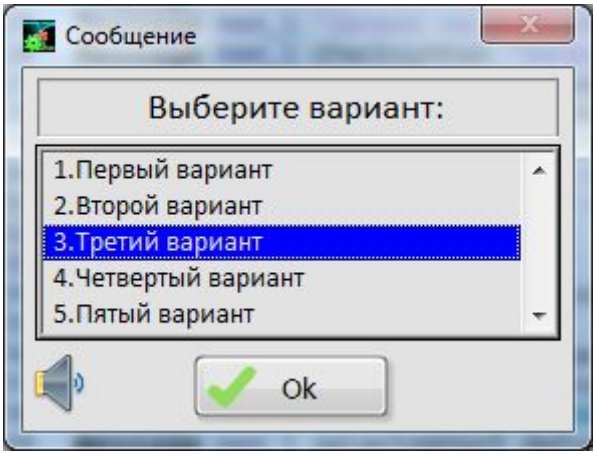


Рис. 4.4. Выбор позиции меню пользователем.

Пример 1:

```
Message mem_5 checksyntax pic=4wireconn.png ”Соедините приборы в соответствии с четырехпроводной схемой соединения, изображенной на рисунке. Введите значение опорного напряжения”

message mem_100 selectmenu=2 defvalue=1 "Проверка остаточной намагниченности: \n1. Не проводилась \n2. Проводилась"
```

Пример 2:

```
Message Mem_1 Выбор набора гирь selectmenu=6 defvalue=1 \n1. Первый \n2. Второй \n3. Третий \n4. Четвертый \n5. Пятый \n6. Шестой
```

Выводит меню (Рис. 4.5.) с набором вариантов:

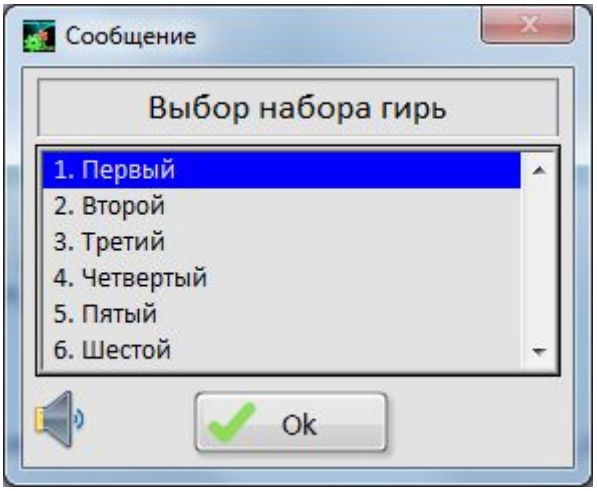


Рис. 4.5. Сообщение с выбором вариантов.

Результат (номер выбора позиции меню - целое число от 1 до 6) помещается в ячейку памяти Mem_1.

В случае загрузки меню из файла параметром LoadMenu в качестве результата в ячейку памяти заносится порядковый номер выбранной позиции, тогда как параметр LoadMenuID сохранит его значение из первой колонки.

Пример 3:

```
message mem_1 LoadMenu = "MessageMenu.csv"     "Выбор варианта"
message mem_1 LoadMenuID = "MessageMenuID.csv" "Выбор варианта"
```

Текстовый файл MessageMenu.csv содержит одну колонку параметров и выбор пользовательского меню сохраняет в ячейке памяти порядковый номер. Файл MessageMenuID.csv состоит из двух колонок. Первая - с параметром, вторая - с его описанием, которое отображается в меню. Значения берутся в кавычки, в качестве разделителя используется символ точки с запятой. Выбор из меню сохраняет в ячейке памяти результат из первой колонки.

4.14. Table - таблица для ввода параметров

Команда предназначена для вывода на экран таблицы, посредством которой пользователь может ввести набор параметров.

Синтаксис:

```
Table параметр <переменная> “Название строки”
<selectmenu=(“a”;”b”;”c”...)> checksyntax <defvalue=num>
```

Описание параметров:

clear - очистка таблицы,

row - описание строки таблицы,

show - отображение таблицы на экране и ввод параметров. Введенные значения сохраняются в указанной переменной в виде массива данных.

<переменная> - ячейка памяти, в которую в виде массива данных помещаются вводимые результаты, указывается только с параметром show,
“Название строки” - текст, отображаемый в строке, обязательно берется в кавычки,
<selectmenu=(“a”;”b”;”c”...)> - формирует пользовательское меню с выбором из списка значений (“a”;”b”;”c”...). Значения задаются в кавычках, в качестве разделителя используется символ точки с запятой,
<defvalue=num> - значение параметра или номер позиции пользовательского меню по умолчанию,
<checksyntax> – опционально, включает проверку корректности введенного числа.

Пример:

```
table clear          # очистка  таблицы
table row "Температура в конце поверки" checksyntax defvalue=20
table row "Влажность в конце поверки" checksyntax defvalue=65
table row "Давление в конце поверки" checksyntax defvalue=1
table row "Время окончания поверки"
table show mem_52    # вывести меню
```

Выводит таблицу (Рис. 4.6.)

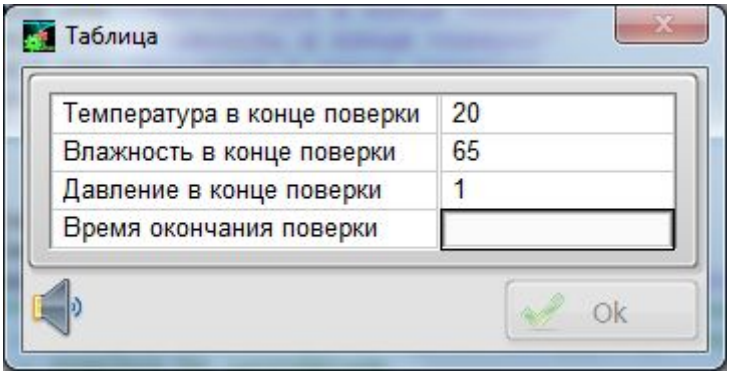


Рис. 4.6. Таблица для ввода параметров.

Введенные значения в виде массива помещаются в ячейку памяти mem_52.

4.15. String - форматирование текста

Команда предназначена для работы со строками, а также форматирования текста, хранимого в ячейках памяти.

Синтаксис:

```
String переменная=<параметр>(переменная)”значение”
```

Описание полей:
переменная - ячейка памяти, в которую помещается результат,
<параметр> - внутренняя команда, выполняющая преобразования текстовой информации,
значение - присваиваемое значение.

Возможные значения параметров:
Upper - преобразование всех букв к верхнему регистру,
Lower - преобразование всех букв к нижнему регистру,
StrSize - устанавливает размер строки в символах,
GetField - делит строку на поля с указанным разделителем и возвращает определенное поле,
Format - форматирует одну ячейку памяти по образцу другой,
TimeFormat - форматирование даты и времени (CurrentTime - текущее значение).

С параметром Format указываются две ячейки памяти, из первой берется значение, а вторая служит опорной для установки формата числа.

Например:

```
math mem_2=99
math mem_1=5m
math mem_3=7
string mem_2 = format(mem_3; mem_1)
```

В результате в ячейке памяти mem_2 сохраняется 7000m (число 7 из ячейки mem_3 форматируется по образцу mem_1 с постфиксом).

При помощи оператора || возможна конкатенация (слияние) строк. Допускается использование множественных вложений параметров.

Синтаксис форматирования даты\времени.
%a - сокращенное название дня недели (например Пт),
%A - полное название дня недели (например - пятница),
%b - сокращенное название месяца (например - сен),
%B - полное название месяца (например - сентябрь),

%c - специфический формат даты и времени, используемый по умолчанию в зависимости от местоположения,
%d - порядковый день месяца (01–31),
%H - час (24-часовой формат) (00–23),
%I - час (12-часовой формат) (01–12),
%j - порядковый номер дня в году (001–366),
%m - порядковый номер месяца (01–12)
%M - минуты (00–59),
%p - флаг AM или PM (после полуночи - до полуночи),
%S - секунды (00–59),
%<digit>u - доли секунды с точностью <digit> знаков,
%U - порядковый номер недели в году (00–53), первым днем первой недели в году считается воскресенье; 00 обозначает первую неделю,
%w - порядковый номер дня недели (0–6), 0 обозначает воскресенье,
%W - порядковый номер недели в году (00–53), первым днем первой недели в году считается понедельник; 00 обозначает первую неделю,
%x - формат даты в зависимости от местоположения,
%.1x - развернутый формат даты,
%.2x - сокращенный формат даты,
%X - формат даты в зависимости от местоположения,
%y - порядковый номер года с столетии (00–99); числа (00–68) представляют год двадцать первого века (2000–2068) а цифры (69–99) представляют год двадцатого века (1969–1999),
%Y - полный формат года (например, 1997),
%z - разница между местным и универсальным временем (ЧЧ:ММ:СС)
%Z - часовой пояс, полностью или сокращенно, в зависимости от настроек операционной системы.

Например:

```
string mem_5=TimeFormat(CurrentTime; %d %B %Y %H:%M)
```

сохранит в ячейке памяти mem_5 текущую дату в формате:
“09 октября 2017 12:50”

В случае присвоения ячейке памяти mem числа в кавычках, число никак не форматируется и сохраняет свой формат. Если присвоить значение без кавычек, число будет отформатировано в наиболее удобный для отображения вариант, например:

```
math mem_1=1,0000000
math mem_2="1,00м"
```

```
math mem_1=1,0000000
          mem_1 = 1,0000000          mem_1 = 1;
math mem_2="2,00м"
 string mem_2="1,00м"           mem_2="1,00м"
math mem_3=mem_1+mem_2
          mem_3 = 1+2,00м          mem_3 = 1.002;
```

Примеры:

```
# длина строки
string mem_1="Hello"
string mem_2 = strsize(mem_1)

# конкатенация (слияние) строк
string mem_2 = mem_1 || " World" || mem_5

# если писать без кавычек, пробелы будут удалены:
string mem_2 = mem_1 ||  World || mem_5

# преобразование всех букв к верхнему регистру
string mem_2 = upper(mem_2)

# преобразование всех букв к нижнему регистру
string mem_2 = lower(mem_2)

# наследование формата
string mem_500 = format(mem_50; mem_5)

# делит строку на поля с указанным разделителем и возвращает
определенное поле
string mem_1 = "Agilent, MSO4503, GO34454522"
string mem_2 = GetField(mem_1;",";3)

# допускается использование нескольких вложений:
string mem_3 = strSize("123" || GetField(mem_1;",";3))
string mem_3 = strSize(GetField(mem_1;",";3))
string mem_3 = strSize(mem_2)
```

Примечание. Переменная mem_1 в данном примере не является массивом, но содержит текстовую строку "Agilent, MSO4503, GO34454522".

4.16. Template - назначение шаблона

Команда предназначена для указания файла специфического шаблона протокола поверки. Имя файла шаблона по умолчанию указывается в настройках администратора. Но файл, указанный непосредственно в скрипте, будет иметь больший приоритет и результаты поверки будут занесены в этот файл шаблона, если он будет присутствовать в папке Temp.

Синтаксис:

```
Template=<название файла>
```

Описание полей:
<название файла> – название файла шаблона Word с расширением doc или docx.

Папка для размещения файла по умолчанию:

```
c:\unitess\temp
```

Пример:

```
Template=template2.doc
```

4.17. Item - переопределение данных для отчета

Команда предназначена для заполнения полей с данными для шапки протокола. Обычно при создании файла отчета в него автоматически вносится ряд стандартных полей, например:
Наименование и обозначение ТНПА или МК: {method1}, {method2}, {method3}
Условия проведения поверки:

- температура окружающего воздуха, ^{0}C {temp}
- относительная влажность воздуха, % {hum}
- атмосферное давление, кПа {pressure}

Выбор эталона, средства калибровки: {etal1}, {etal2}, {etal3}.
Данные специалиста, проводившего поверку:

- должность {position}
- имя {name}

Итоговое заключение результата поверки {test_res} о соответствии требованиям.
Присвоить или переопределить значения всех этих параметров можно с помощью команды **Item**.

Синтаксис:

```
Item {<название поля>} <значение поля>
```

Описание полей:
<название поля> – название поля в шаблоне скрипта, в которое необходимо подставить требуемое значение. Название поля всегда указывается в фигурных скобках,
<значение поля> – число или текст, который необходимо подставить в шаблон по месту указанного поля.

UniTesS APM может импортировать в протокол любую информацию из базы данных посредством хранимой процедуры: SP_GET_DPRT_ALL_INFO
В меню “**Project**” --> “**Данные об образце**” можно увидеть набор доступных для текущего скрипта полей. Хранимая процедура возвращает имя поля, описание и значение для текущего образца. При создании отчета UniTesS APM ищет имена полей в шаблоне протокола и производит замену на их значения.

Также существуют предопределенные системные поля с псевдонимами:
{testing_result} - Глобальный результат выполнения скрипта. Вместо этого псевдонима UniTesS APM подставит фразу о положительном или отрицательном результате выполнения скрипта, причем фраза будет взята для конкретного отдела текущего сотрудника из таблицы S_APPROVED.
{user_full_name} - Фамилия И.О. текущего пользователя из таблицы SYS_USERS.
{user_position} - полное название должности для текущего пользователя из таблицы S_JOB.

{user_department} - полное название отдела для текущего пользователя из таблицы S_DEPARTMENT.
{current_time} - текущая дата и время в формате в соответствии с настройками операционной системы.
{start_time} - время начала тестирования в формате в соответствии с настройками операционной системы.
{sw_ver_apm} - текущая версия ПО UniTesS APM.
{sw_ver_db} - текущая версия базы данных.
{scr_name} - имя текущего скрипта.
{scr_date} - дата последней модификации текущего скрипта в базе данных.
{scr_ver} - версия скрипта, берется из текста скрипта командой ScriptVersion.
{scr_uhesh} - контрольная сумма скрипта. Рассчитывается без учета пробелов, табуляций, символов переноса строки, комментариев, item.

Можно задать формат даты и времени для полей {current_time} и {start_time}

Например:

```
{current_time%<"%d" %B %Y  %H:%M>T}
```

формирует в протоколе строку

```
"27" октября 2017  14:48
```

Правила форматирования даты/времени приведены в описании команды **String**.

Пример:

```
item {model} Tektronix TDS 3052C
item {method} МП-179/447-2006, "Мультиметры цифровые MS822G. Методика поверки".
item {calibrator} МНИПИ Н4-101
item {temp} 20
item {hum} 65
item {pressure} 99
```

4.18. Report - формирование отчета

Команда предназначена для добавления значений в отчет в соответствии с методом измерений, к которому относится закладка в файле шаблона протокола. Названия закладок, методы и расшифровка полей хранятся в базе данных. Все значения, указанные в данной команде, вносятся по месту закладки в шаблон Word. Закладка может обозначать таблицу целиком, находиться внутри или вне ее. По умолчанию в программе предусмотрено три метода занесения данных в отчет: **string**, **table** и **row**. Первый метод заносит отдельную строку по месту закладки в шаблоне Word. Второй метод заносит строку значений, разделенных пробелами или символами табуляции, в таблицу, обозначенную закладкой. При формировании отчета каждая строка будет добавляться в конец таблицы в отчете. Для присвоения закладки таблице необходимо выделить всю таблицу, нажав на символ  в левом верхнем углу этой таблицы, затем добавить закладку. Количество полей в строке команды Report должно совпадать с количеством ячеек в строке таблицы в отчете. Если количество полей в командной строке окажется больше числа ячеек в строке таблицы, все избыточные значения будут занесены в последнюю ячейку таблицы, каждое в новой строке. После заполнения всех таблиц программа ищет таблицы, оставшиеся пустыми, и удаляет их вместе с заголовками, которые были выделены закладкой, соответствующей таблице. При добавлении новой таблицы в шаблон рекомендуется оставлять пустую строку после предшествующего текста, затем следует строка с названием таблицы и сама таблица.

Для работы команды **Report** необходимо определить **тип данных** и указать их формат. Это можно сделать в меню ***Администрирование*\ *Автоматизация измерений* \ *Типы данных для протокола***. Также необходимо добавить **шаблон протокола** в меню ***Администрирование*\ *Автоматизация измерений* \ *Шаблоны протоколов*** и там же добавить поля для шапки протокола (опционально). При добавлении скрипта необходимо связать его и используемым типом данных и шаблоном протокола.

Все измерения в АРМ UniTesS закономерно заканчиваются командой **Report**, которая формирует таблицы результатов в базе данных и готовит их для отправки в файл отчета.

Синтаксис:

```
Report <название закладки> <значение поля> <значение поля>...
```

Описание полей:
<название закладки> – название закладки в шаблоне Word, по месту которой необходимо занести значение,

<значение поля> – данные для занесения в отчет. Поле может быть ячейкой памяти, текстовой строкой, либо представляться численным значением. Значение, введенное в кавычках (например “abc”) будет определено как строка и добавлено в отчет без преобразований, как есть. Для занесения данных в таблицу или в произвольное место отчета, значения должны быть разделены как минимум одним символом пробела или табуляции.

Пример 1:

```
Report FRG mem_1 3,45 10k 4,1% mem_10
Report NAM "Иван Иванович Иванов"
```

Для того, чтобы команда **Report** отработала корректно и все нужные данные были добавлены в Протокол, необходимо соблюдение нескольких условий. Рассмотрим на примере измерения напряжения постоянного тока.

1. В шаблон протокола добавляется таблица для занесения результатов измерений.

- Название таблицы: “Определение погрешности измерения напряжения постоянного тока”
- Количество колонок: **шесть**
- Наименования колонок и соответствующих параметров:
    - “Предел измерений, В”
    - “Поверяемая точка №, В”
    - “Результаты поверки, В”
    - “Погрешность, ед. мл. разряда ±Δ”
    - “Предел основной допускаемой погрешности, ед. мл. разряда ±Δ”
    - “Вывод о соответствии”

Всей таблице, включая ее название, присваивается метка: **VDC**

2. Создается тип данных с названием **VDC**

Метод занесения: **table**
Включает **шесть** параметров:

- Предел измерения, В — формат - %,;%.0dw
- Поверяемая точка, В — формат - %Reference_w
- Результат поверки, В — формат - %,;%.0dw
- Погрешность, В — формат - %,;%.0dw
- Предел допускаемой осн. погрешности, В — формат - %,;%.0dw
- Вывод о соответствии — формат - %s10.

3. В скрипте описывается функция с названием **VDC**

```
# * Метод определения погрешности измерения постоянного напряжения
*
```

```
#F/ Результат; Диапазон, В; Поверяемая точка, В; Погрешность, %;
Погрешность, ед.; Разрешение
function  VDC  mem_2  mem_3  mem_4  mem_5  mem_6
```

Которая заканчивается вызовом команды **Report** с **шестью** аргументами:

```
#Отчет: Предел,| Повер  | Результат | Погр. | Предел доп. | Вывод
#          В   | точ, В | измер, В  | ед.мл.| погр. ед.мл.|
report VDC  mem_2  mem_3  mem_1  mem_8  mem_7  mem_12
EndFunction
```

Метка таблицы **VDC** в шаблоне протокола, название типа данных для шаблона протокола **VDC** и аргумент команды Report **VDC** должны совпадать и соответствовать по числу параметров. Название вызываемой функции (в нашем примере тоже VDC) может отличаться и выбирается произвольно.

Итак, основные требования, которые обязательно должны соблюдаться:

- Название **метки**, назначаемой таблице результатов в шаблоне протокола, название **типа данных** и первый аргумент команды **Report** должны быть идентичны.
- Количество колонок в таблице протокола для занесения результатов, число параметров типа данных и количество аргументов команды **Report** должны совпадать.
- Формат и местоположение аргументов должны соответствовать по названию и типу.

Работа с несколькими Report по одному параметру

Каждый **Report** и результат сравнения **Compare** подписывается временем выполнения и хешем. Этот механизм необходим для соотнесения дерева (просмотр **Simple View**) и скрипта. Каждый элемент дерева подписывается определенным хешем. Хеш рассчитывается из аргументов функции. Если в теле функции выполняются команды **Report** и **Compare**, то их результат помещается в базу данных и подписывается хешем функции и таким образом ассоциируется с определенным элементом дерева. Когда АРМ выполняет **Report** или **Compare**, по хешу проверяется наличие данных (старые измерения) и время выполнения. В конечном итоге АРМ оставляет только свежие данные.
Для того, чтобы этот механизм работал, необходимо чтобы все команды **Report** и **Compare** находились в теле подписанной функции.

Примеры работы со статическими uhesh #uhesh (heshexampl_tree)

```
call RFSAAttLevelErr 10 10 4 #uhesh(heshexampl1) первый подписанный
вызов функции с явным хешем
```

```
call RFSAAttLevelErr 20 20 4 #uhesh(heshexampl2) второй подписанный
вызов функции с явным хешем
call RFSAAttLevelErr 30 20 4 #uhesh(heshexampl3) третий подписанный
вызов функции с явным хешем
```

Также можно запрашивать состояние дерева и **Report** по хешу:
IsTreeItemEnabled - выдает 1 если элемент дерева выполняется,
IsReportExist - выдает 1 если **Report** существует,
IsCompareExist - выдает 1 если результат **Compare** существует,
IsComparePositive - выдает 1 если результат **Compare** положителен,
IsReportPositive - действует аналогично IsComparePositive.

Пример 2:

```
math mem_1=IsTreeItemEnabled(heshexampl1)
math mem_1=IsReportExist(heshexampl1)
math mem_1=IsCompareExist(heshexampl1)
math mem_1=IsComparePositive(heshexampl1)
math mem_1=IsReportPositive(heshexampl1)
   if IsTreeItemEnabled(heshexampl1)
    delay 100
   endif
if IsReportExist(heshexampl1)
 delay 100
endif
   if IsCompareExist(heshexampl1)
    delay 100
   endif
if IsComparePositive(heshexampl1)
 delay 100
endif
   if IsReportPositive(heshexampl1)
    delay 100
   endif
```

Чтобы обратиться к определенному элементу дерева, необходимо знать его хеш. Так как хеш подписанной функции вычисляется динамически при загрузке скрипта исходя из аргументов функции, то проще всего задать его статически в комментарии к вызову функции с помощью директивы
#uhesh(heshexampl1), где heshexampl1 - строковое значение уникального хеша.

Команда **TreeRules** устанавливает правила работы с деревом (Tree, simple view). Это декларативная команда, она не выполняется. Используется если необходимо, чтобы пользователь не смог отключить определенные элементы дерева или при отключении одного элемента, автоматически отключались и другие. Например:

```
# пункты всегда включены
TreeRules Fixed heshexampl_tree1;
# связанные пункты. При отключении\включении одного, другие тоже
# отключаются\включаются
TreeRules Linked heshexampl1; heshexampl2; heshexampl3

# может быть любое количество групп или связанных пунктов
TreeRules Linked heshexampl_l1; heshexampl_l2
```

Допустимо указывать только латинские буквы, цифры и знак нижнего подчеркивания. Если APM встретит одинаковые хеши, то они проиндексируются добавлением символа “**!**” (восклицательный знак). Как только uhesh обозначен для функции или комментария, появляется возможность управлять элементами дерева с помощью команд:

```
Tree Enable  heshexampl1; heshexampl2; heshexampl3
Tree Disable heshexampl1; heshexampl2; heshexampl3
```

Можно управлять целым диапазоном хешей

```
Tree Enable  heshexampl1:heshexampl3
Tree Disable heshexampl1:heshexampl3

Tree Disable heshexampl1; heshexampl2
Tree Disable heshexampl3
Tree Enable  heshexampl1:heshexampl3
Tree EnableALL                  # выполнить все пункты
Tree EnableOnlyUncompleted      # выполнить только невыполненные.
```

Пример 3:

```
#/ Тестирование управления деревом  #uhesh(heshexampl_tree)
call VDC         0,001          4     #uhesh(heshexampl1)    первый
#подписанный вызов функции с явным хешем
call VDC         0,0011         4
call VDC         0,0012         4
call VDC         0,0013         4
call VDC         0,002          4       #uhesh(heshexampl2)    второй
#подписанный вызов функции с явным хешем
call VDC         0,0014         4
call VDC         0,0015         4
call VDC         0,0016         4
call VDC         0,003          4       #uhesh(heshexampl3)
#последний подписанный вызов функции с явным хешем
```

Пример 4:

```
math mem_1=IsTreeItemEnabled(heshexampl_tree)
math mem_2=IsTreeItemEnabled(heshexampl1)
```

4.19. SelectReport и DeleteReport - операции с отчетами

С помощью команды **SelectReport** все данные, отправленные в протокол командой **Report**, можно выгрузить в одну ячейку памяти **mem_** в виде массива.
Для этого в типах данных необходимо ввести краткое обозначение колонки. Например **range** для Диапазона измерений, **point** для Измеряемой точки и **result** для Результата измерений.
Теперь можно одним вызовом выгрузить всю колонку результатов **POINT** в ячейку mem_5 для параметра **VDC**. При этом в ячейку mem_5 помещается массив результатов.
В описании формата данных для протокола (Рис. 4.7.) Обозначение параметра вводится как **#Tag**.

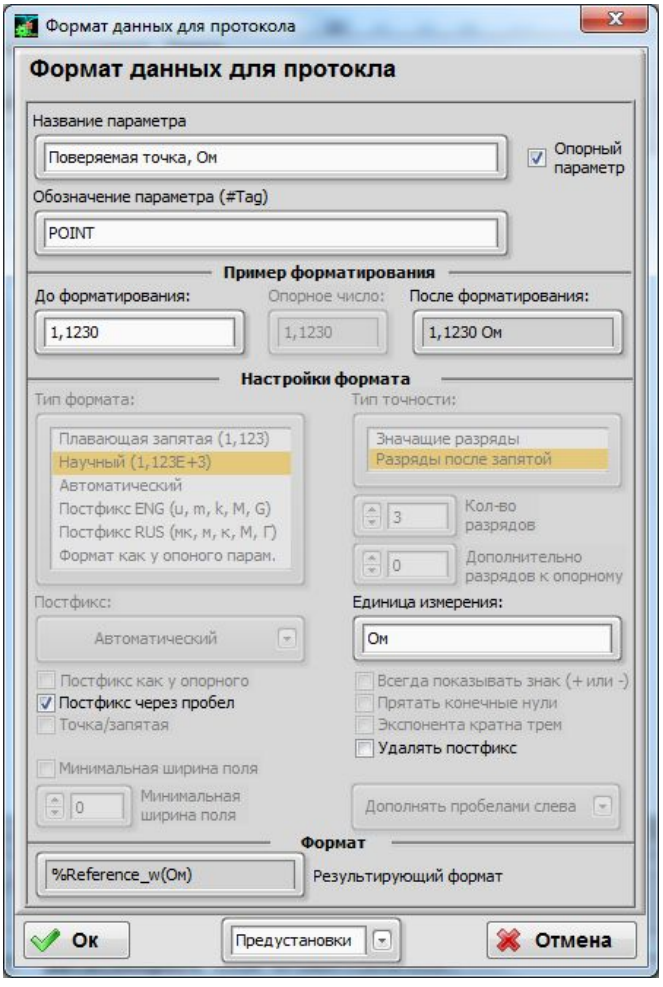


Рис. 4.7. Формат данных для протокола.

После чего в описании данных для протокола появляются соответствующие метки, например (#RANGE), (#POINT) и (#RESULT):

```
data_description VDC table "Погрешность измерения напряжения
постоянного тока"; "Предел измерения, В (#RANGE):%,;%#1.0dw(B)";
"Поверяемая точка, В (#POINT):%Reference_w(B)"; "Результат поверки,
В (#RESULT):%,;%1.0dw(B)"; "Погрешность, В:%,;%1.0dw(B)"; "Предел
допускаемой осн. погрешности, ед. мл. разряда:%,;%1.0dw(B)"; "Вывод
о соответствии"
```

Синтаксис:

```
SelectReport <тег> as <ячейка памяти> from <метка>
```

Описание полей:

<тег> - обозначение параметра,
<метка> - метка, описывающая формат данных для протокола, а также итоговую таблицу результатов в протоколе, аргумент команды **Report**,
<ячейка памяти> – ячейка памяти mem, в которую помещается массив данных.

Пример.

```
SelectReport result as mem_5 from VDC;
```

Можно вводить различные условия, делая выборку с помощью ключа **where**:

```
SelectReport result as mem_5 from VDC where (range = 200) and (point = 20);
```

```
# в отчет: Рез. измер;| Предел, В;| Повер. точка, В;| Доп. осн. погр.
report VDC mem_1  200 20 0.2 Mem_10
```

Все данные, отправленные в протокол командой **Report**, можно удалить командой **DeleteReport**. Для этого в типах данных также используется короткое обозначение колонок например range, point, result.

Можно вводить разнообразные условия с помощью ключа where:

```
DeleteReport from VDC where (range = 10) and (point = 200);
```

Удаление всей таблицы:

```
DeleteReport from VDC;
```

5. Работа с драйверами

Драйвер служит посредником при передаче команд и получением информации между выполняемой программой (скриптом) и прибором, например поверяемым средством измерения или калибратором.

5.1. Управление приборами

ПО UniTesS APM реализует несколько подходов к управлению оборудованием по интерфейсам. Существует четыре варианта:

- управление непосредственно командами прибора через *VISA интерфейс*;
- управление через персональный *UniTesS драйвер*;
- управление через *IVI драйвер;*
- управление через плагин.

При помощи специального модуля UniTesS Vision обеспечивается возможность производить поверку и комплексную автоматизацию измерений для приборов, не имеющих аппаратно-программного интерфейса. Работа модуля Vision описана в разделе 12 данного Руководства.

UniTesS APM может управлять оборудованием через следующие интерфейсы ПК:

- USB,
- Ethernet,
- GPIB,
- RS232 и других стандартов.

Для подключения по GPIB (General Purpose Interface Bus, интерфейсная шина общего назначения) необходимо использовать преобразователь интерфейсов USB-GPIB. UniTesS APM поддерживает работу со всеми видами преобразователей от National Instruments, а также Prologix USB-GPIB.

VISA (Virtual Instrument Software Architecture) — широко используемый стандартизированный интерфейс ввода-вывода в области тестирования и производства измерений для управления приборами с персонального компьютера.

Посредством VISA-интерфейса значительно упрощается управление любым прибором, имеющим возможность подключения к компьютеру и документированную систему команд. В случае использования команд прибора достигается наибольшая гибкость в управлении, но вместе с тем иногда теряется универсальность, так как набор команд у каждого производителя, а иногда и у различных приборов одного производителя, может отличаться.

Для настройки интерфейса взаимодействия с конкретным прибором необходимо в скрипте описать его конфигурацию с помощью специальных команд настройки: **DriverSet**, **PortConfig**, **IVIConfig, VisionSet**. Команды конфигурации назначают устройствам персональные имена (псевдонимы) для использования внутри скрипта – *alias*. Это удобно не только для наглядности и читаемости текста скрипта, но также позволяет работать с несколькими одинаковыми приборами, целыми классами приборов, совместимыми в плане системы команд. При работе через персональный драйвер устройства процедура может значительно усложниться, так как возникает необходимость создавать отдельный скрипт для каждого прибора.

Специализированные *UniTesS драйвера* разрабатываются для уникального, сложного оборудования или в случаях, когда необходимо реализовать специфический функционал. При работе через персональный *UniTesS драйвер* выполнять дополнительные настройки или конфигурирование в скрипте не требуется, все необходимые параметры хранятся в драйвере, это значительно упрощает задачу. Перечень *UniTesS драйверов* ограничен, но постоянно расширяется. Использование персональных *UniTesS драйверов* предпочтительно, так как их код оптимизирован для конкретных моделей приборов, передача команд и получение информации происходит напрямую, гораздо быстрее и надежнее.

С целью упрощения взаимодействия с некоторыми классами приборов, оптимизации и ускорения разработки автоматизированных рабочих мест, в UniTesS АРМ внедрена поддержка IVI-драйверов.
IVI-драйверы – это драйверы для управления приборами, которые обладают повышенной производительностью и гибкостью, дают возможность пользователю создавать более сложные приложения для производства измерений, требующих универсальности, взаимозаменяемости или возможности симуляции оборудования. Для достижения взаимозаменяемости IVI-фондом, который занимается разработкой и поддержкой таких драйверов, были определены спецификации для следующих классов приборов:

- осциллографы (Scope);
- цифровые мультиметры (DMM);
- генераторы сигналов (FGen);
- источники питания постоянного тока (DC Pwr Supply);
- коммутаторы (Switch);
- измерители мощности (Power Meter);
- генераторы радиочастотных сигналов (RF Sig Gen);
- анализаторы спектра (Spec An);
- счетчики (Counter);
- понижающие преобразователи (Downconverter);
- повышающие преобразователи (Upconverter);
- дигитайзеры (Digitizer);
- источники питания переменного тока (AC Pwr Supply).

Более подробно способы и команды управления приборами с использованием IVI драйверов описаны в разделе 10.

5.2. DriverSet - управление прибором

(Универсальный драйвер-интерфейс и UniTesS драйвер)

Команда предназначена для передачи управляющих команд в прибор через персональный драйвер устройства.

Синтаксис:

```
DriverSet <имя драйвера> <строка команды>
```

Описание полей:
<имя драйвера> – имя используемого драйвера.
<строка команды> – строка команды прибора, индивидуальная для каждого драйвера и прибора. Команда может содержать наименование параметра и устанавливаемое значение.

Формат:

```
<команда>=<допустимые параметры>;<команда>=<допустимые
параметры>...
```

Допускается ввод набора команд с параметрами, в качестве разделителя используется символ точки с запятой.

Пример:

```
DriverSet 9100 IDC=50m  aux
DriverSet Fluke8508A set=filt_on
DriverSet B1_28 mode=meas
DriverSet B1_18 mode=gen
DriverSet N4_101 Stop
DriverSet N4_101 Timeout=10000
DriverSet f12x SetMeas=VDC; HorizontalRange=100u;
VerticalRange=10m; Voffset=-30m; TrigLevel=20m;
```

5.3. DriverGet - получение информации

(Универсальный драйвер-интерфейс и UniTesS драйвер)

Драйвер служит посредником при передаче команд и получением информации между выполняемой программой (скриптом) и прибором.

Команда предназначена для чтения параметра из прибора через персональный драйвер устройства и запись его в ячейку памяти.

Синтаксис:

```
DriverSet <имя драйвера> <ячейка памяти> = <имя параметра>
```

Описание полей:

<имя драйвера> – имя используемого драйвера.
<ячейка памяти> – имя одной из внутренних ячеек памяти ПО (mem_1, mem_2, …), в которую будет помещен ответ.
<имя параметра> – наименование параметра, считанного из устройства. Набор параметров индивидуален для каждого драйвера и прибора.

Пример:

```
DriverGet 9100  mem_1 = period;
DriverGet f12x  mem_1 = Main_A;
DriverGet B1_28 mem_1 = value;
DriverGet B1_18 mem_1 = value;
DriverGet Fluke8508A mem_1 = read;
```

Подробное описание и параметры команд для различных приборов приведены в Приложениях:
Приложение 10 - драйвер осциллографа Fluke 12x.
Приложение 11 - драйвер калибратора - вольтметра B1-28.
Приложение 12 - драйвер вольтметра B1-18.
Приложение 13 - драйвер вольтметра Fluke 8508A.
Приложение 14 - драйвер калибратора Fluke серий 55xx и 57xx.

6. Работа с IVI-драйверами

Драйвера класса IVI (Interchangeable Virtual Instruments - Взаимозаменяемые Виртуальные Инструменты) - это стандартизированные драйвера контрольно-измерительных приборов, позволяющие заменять одно устройство другим незаметно для основного приложения. Разработкой и поддержкой драйверов IVI занимается компания National Instruments. В скрипте приборы одного класса управляются одинаковыми командами.

Внимание! После установки драйвера следует запустить приложение NI - Measurement & Automation Explorer для его настройки.

Перечень классов IVI

- осциллографы (Scope);
- цифровые мультиметры (DMM);
- генераторы сигналов (FGen);
- источники питания постоянного тока (DC Pwr Supply);
- коммутаторы (Switch);
- измерители мощности (Power Meter);
- генераторы радиочастотных сигналов (RF Sig Gen);
- анализаторы спектра (Spec An);
- счетчики (Counter);
- понижающий преобразователь (Downconverter);
- повышающий преобразователь (Upconverter);
- дигитайзеры (Digitizer);
- источники питания переменного тока (AC Pwr Supply).

Описание управляющих команд осциллографов (Scope) приведено в Приложении №6.

6.1. IVIConfig - конфигурация IVI-интерфейса

Команда предназначена для конфигурирования и инициализации IVI-интерфейса устройства. IVI имеет унифицированный интерфейс для каждого класса приборов. Например, для всех осциллографов, работающих через IVI-драйвер, синтаксис команд и названия параметров будет одинаковым.

Синтаксис:

```
IVIConfig <имя> [<IVI-класс>,<драйвер>,<model= или asset= >]
```

Описание полей:
<имя> – имя, назначаемое прибору для использования в скрипте.
<IVI-класс> – IVI-класс устройств.
Допустимые значения, в зависимости от класса устройств по спецификации IVI: Scope, DMM, Fgen, DCpwr, Switch, Powermeter, RFsiggen, ACPwr, Counter, Downconverter, Upconverter, Digitizer, Specan.
<драйвер> – наименование используемого IVI-драйвера. Перечень IVI-драйверов, установленных на компьютере можно найти в приложении Measurement and Automation Explorer.
<модель> – модель устройства, с которой работает выбранный драйвер.

Пример:

```
IVIConfig scope  [scope,Ag546xx_my,model=OSO7104A или asset=TCPO:
…]
```

6.2. IVIdriverSet - управление приборами

Команда предназначена для передачи команд в прибор через IVI-драйвер устройства.

Синтаксис:

```
IVIdriverSet <имя> <команды IVI>
```

Описание полей:

<имя> – имя, назначенное прибору для использования в скрипте.
<команды> – строка, содержащая команды, индивидуальные для конкретного IVI-класса приборов.

Перечень команд, доступных для класса осциллографов “Scope” с допустимыми параметрами находится в Приложении №6 данного руководства.

Пример:

```
IVIdriverSet scope Channel=2; ChannelEnabled=ON
```

6.3. IVIdriverGet - чтение параметров

Команда предназначена для чтения параметров из прибора через IVI-драйвер устройства.

Синтаксис:

```
IVIdriverGet <имя> <ячейка памяти>=<имя параметра>, <ячейка памяти>=<имя параметра>,…
```

Описание полей:

<имя> – имя, назначенное прибору для использования в скрипте.
<ячейка памяти> – имя одной из внутренних ячеек памяти ПО (mem_1, mem_2, …), в которую будет помещен ответ.
<имя параметра> – параметры, индивидуальные для определенного IVI-класса приборов.

Пример:

```
IVIdriverGet scope mem_1 = period; mem_2 = frequency
```

7. Работа с портом через VISA

ПО UniTesS АРМ может работать с прибором через VISA интерфейс.

7.1. PortConfig - конфигурация порта

Настройка порта для коммуникации с прибором посредством его внутреннего интерфейса осуществляется командой **PortConfig**.

Синтаксис:

```
PortConfig <alias> [<общие настройки>] <название интерфейса>
[<индивидуальные настройки>]
```

Описание полей:

<alias> – имя устройства, принятое для использования в скрипте.
<общие настройки> – настройки таймаута и символ признака конца строки, являющиеся общими для любого типа интерфейса. Записываются в квадратных скобках в следующем виде: [<таймаут, мс>, <признак конца строки>].
<название интерфейса> – тип интерфейса, на котором подключен прибор. Допустимые значения: COM, USB, GPIB, Ethernet.
<индивидуальные настройки> – специфические настройки для определенного типа интерфейса.

Для COM:

[<имя порта>, <скорость, бод>, <битов в слове>, <стоповые биты>, <четность>, <управление потоком>].
Допустимые значения:
<имя порта>: com1, com2 и т.д.
<скорость, бод>: 9600, 19200, 57600, 115200
<битов в слове>: 7, 8
<стоповые биты>: 1, 1.5, 2.
<четность>: odd, even, mark, space.
<управление потоком>: dtr/dsr, cts/rts, xon/xoff.

Для USB:
[<модель прибора>] - Модель необходимо взять из настроек программы Measurement and Automation Explorer.

Для GPIB, Ethernet:
[<alias>].

Подключение необходимо предварительно настроить в приложении Measurement and Automation Explorer, указав конкретный alias для устройства.

Пример:

```
PortConfig <alias> [3000,\n] com [com3, 9600, 8, 1, none, dtr/dsr]
PortConfig <alias> [1500,\n] USB [simple device]
PortConfig <alias> [1500,\n] GPIB [gpib0::18::instr]
PortConfig <alias> [1500,\n] Ethernet [tcpip:192.168.0.1]
```

7.2. PortWrite - запись в порт

Запись команды прибора в порт через VISA.

Синтаксис:

```
PortWrite <alias>  <команда>
```

Описание полей:
<alias> – имя устройства, принятое для использования в скрипте.
<команда> – команда или последовательность команд. Набор команд для каждого прибора индивидуален.

Пример:

```
PortWrite THS3024  *idn?
```

7.3. PortRead - чтение из порта

Чтение данных, ответа от прибора из порта.

Синтаксис:

```
PortRead <alias> <ячейка памяти> <номер поля> [<разделитель>]
```

Описание полей:

<alias> – имя устройства, принятое для использования в скрипте.
<ячейка памяти> – имя одной из внутренних ячеек памяти ПО, в которую будет помещен ответ.
<номер поля> – порядковый номер поля из строки ответа прибора. Параметр может быть пропущен. В этом случае в указанную ячейку памяти будет записана вся строка ответа от 1 до n.
<разделитель> – символ, разделяющий поля в строке ответа, например со специализированными драйверами.

Пример:

```
PortRead THS3024 mem_3 1 [,]
```

8. Использование приложения Vision

Существует множество приборов, которые не имеют портов ввода-вывода для подключения к компьютеру. Специальное приложение UniTesS Vision (Рис. 8.1.) позволяет выполнять поверку приборов, которые не имеют аппаратно-программного интерфейса управления. Показания приборов с индикаторами любого типа считываются посредством видеокамеры, обрабатываются и передаются ПО UniTesS.

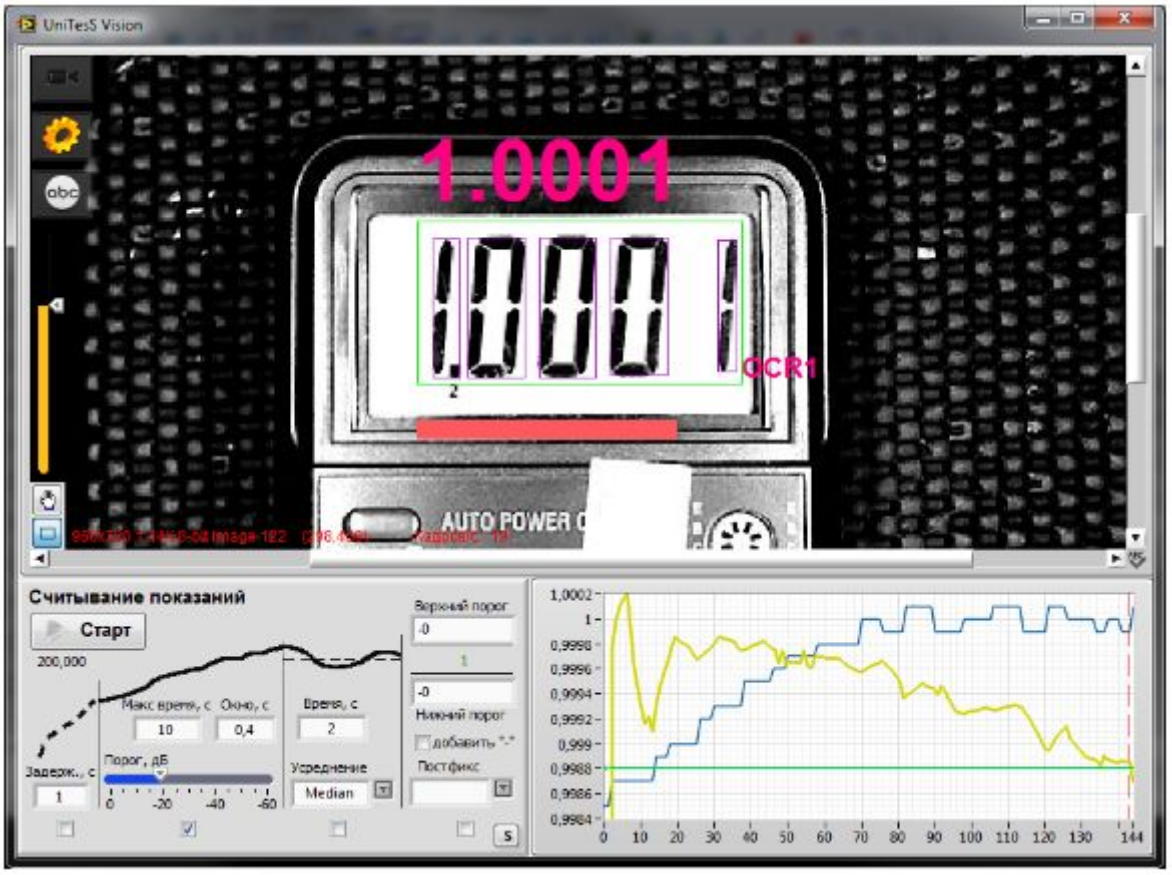


Рис. 8.1. Приложение Vision.

Процесс считывания показаний проходит в четыре этапа:

- устанавливается задержка;
- выполняется алгоритм сходимости;
- проводится математический анализ (RMS, Peak, Average);
- подтверждение результата.

Алгоритм сходимости ожидает установления показаний. Медианный фильтр выбирает значение, наиболее часто отображаемое на экране. При выходе за порог выводится сообщение с просьбой подтвердить результаты.

Алгоритм распознавания приложения Vision обеспечивает возможность обучения и успешного считывания показаний практически с любых приборов с цифровыми индикаторами.

8.1. VisionSet - настройка параметров

Команда выполняет настройку разрешения камеры, алгоритма распознавания символов и проверки соответствия, процесса измерения, сегментации, специальные настройки знака минус, а также устанавливает формат значений для результата. Все настройки вступают в силу только после команды UpdateOCR.

Синтаксис:
```
VisionSet параметр=значение;параметр=значение;параметр=значение...
```

Описание полей:
параметр – наименование параметра настройки,
значение – значение параметра.

Описание параметров:
Resolution - разрешение и формат видео,
OCRFile - параметры распознавания,
MinCharSize - минимальный размер символов,
MinCharWidth - минимальная ширина символов,
MinCharHeight - минимальная высота символов,
MaxCharSize - максимальный размер символов
MaxCharWidth - максимальная ширина символов,
MaxCharHeight - максимальная высота символов,
MinCharSpace - минимальный промежуток между символами,
MaxHorElemSpace - максимальный горизонтальный промежуток между элементами одного символа,
MaxVertElemSpace - максимальный вертикальный промежуток между элементами одного символа,
LowerLimit - нижний предел измерения,
UpperLimit - верхний предел измерения,
NumberOfErrosions - количество дефектов отображения,
FormatExample - пример формата результата,
RemoveParticlesTROI - удалить частицы, касающиеся ROI,
ThrsMode - устанавливает режим (например Uniform или др.),
ThrsLimits - диапазон для заданного режима,
DelayEn - (yes/no) задержка (есть/нет), включает/отключает опцию задержки (dis/en),
MeasDelay - задержка,
ConvEn - (En/Ru) включает/отключает опцию установки времени распознавания,
AnalysisEn - (En/Ru) включение опции времени анализа,
AnalysisType - тип анализа (MEDIAN),
AnalysisTime - время анализа,

Threshold - порог, единицы младшего разряда,
Window - окно (время распознавания в окне),
MaxTime - максимальное время измерения,
AcceptanceLevel - уровень восприятия,
DarkChar - (yes/no) затемнение символов (есть/нет),
Postfix - (yes/no) наличие постфикса (есть/нет),
Minus - (yes/no) присутствие знака минус (есть/нет).

Пример:

```
VisionSet Resolution=1280x720 MJPG 30.00fps
VisionSet OCRFile=OCR1;
VisionSet MinCharSize=10;  MinCharWidth=3; MaxCharWidth=200;
MinCharHeight=3; MaxCharHeight=200;
VisionSet MinCharSpace=1; MaxHorElemSpace=0; MaxVertElemSpave=20;
VisionSet ThrsMode=Uniform; AcceptanceLevel=600; DarkChar=yes;
NumberOfErrosions=0; RemoveParticlesTROI=no; ThrsLimits=0 220;
VisionSet Minus=no; MinusHoffset=0; MinusVoffset=0; MinusHeight=20;
MinusWidth=70
VisionSet UpdateOCR
VisionSet DelayEn=no; MeasDelay=0,5; ConvEn=En; Threshold=2;
Window=2; MaxTime=10;
VisionSet AnalysisEn=En; AnalysisType=MEDIAN; AnalysisTime=1;
SetMinus=No; Postfix=no
VisionSet CheckResult=En; LowerLimit=0,000; UpperLimit=1,000;
VisionSet FormatExample=200,000
```

8.2. VisionGet - считывание показаний

Команда выполняет считывание показаний с прибора и помещает значение в указанную ячейку памяти.

Синтаксис:

```
VisionGet <ячейка памяти>=read
```

Описание полей:
<ячейка памяти> – имя одной из внутренних ячеек памяти ПО, в которую будет помещено считанное с прибора значение,

Пример:

```
VisionGet mem_1=read
```

9. Работа с плагинами

С целью исключения рисков, связанных с вероятным повреждением средств измерения, приборов, сложного и дорогостоящего оборудования, некоторые эксперименты проводятся с так называемыми Виртуальными Приборами (ВП или VI), которые представляют собой не реальный прибор, а программу, исполняемый файл с расширением exe. Такая программа обеспечивает полную имитацию работы реального прибора, управляется аналогичным набором команд (Control) и на выходе выдает определенный набор параметров, отслеживаемый благодаря индикаторам состояния (Event).
С помощью UniTesS Plugin Link можно подключать к UniTesS APM любой ВП. Идея состоит в том, что подключая библиотеку UnitessPluginLink.llb к вашему ВП, прямо из UniTesS APM вы можете командами менять значения управляющих параметров (Control) и получать результат, генерировать при этом нужное событие (Event), если это необходимо.
UniTesS Plugin Link может самостоятельно генерировать файл с описанием всех доступных команд и аргументов.

Последовательность действий по подключению UnitessPluginLink.llb к вашему ВП.

- Создаете ваш ВП. Сначала добейтесь корректной работы, и только после этого подключайте UnitessPluginLink.llb
- Помещаете его в проект UnitessPluginLink.llb
- При инициализации вашего ВП с помощью UPL Server Regist Control.vi регистрируйте параметры (Control), которыми вы хотите управлять и генерировать события
- Далее выполняйте UPL Server RUN.vi с названием создаваемого сетевого сервиса.

ВНИМАНИЕ: название сервиса чувствительно к регистру.

Теперь можно управлять ВП из UniTesS APM с помощью команд:

```
PluginConfig ServiceName IP="192.168.0.1"  # Конфигурация сетевого подключения
PluginConfig ServiceName IP=localhost      # Конфигурация сетевого подключения
PluginSet ServiceName ControlName=12 Event # изменение Control и Event
PluginSet ServiceName ControlName          # Просто событие без смены значения
PluginGet ServiceName mem_1=ControlName
```

По умолчанию для каждого плагина есть набор системных команд:

```
PluginSet ServiceName timeout = 5000 # Очень важно увеличить время
#ожидания отклика если вы используете Event.
PluginSet ServiceName lock; unlock   # включает и отключает
#возможность пользователей управлять Control.
PluginSet  help  # создаем и открываем файл справки
```

В файле справки будут перечислены все доступные Control с доступными аргументами и описанием из tip strip. Но этот файл справки необходимо редактировать вручную, например, если вашим ВП используются динамически изменяемые списки enum list и т.д.

Если вы в APMe для Control использовали определенный Event, то в вашем ВП также будет генерироваться аналогичный Event для Control и **очень важно** подтвердить завершение действия с помощью UPL Server Event DONE.vi
Это необходимо в случае, когда ваш Event только запускает некий процесс и нужно дождаться завершения и передать в APM кластер ошибок.

При выходе из программы необходимо закрыть сервис UPL Server STOP.vi

При старте скрипта APM автоматически загружает все используемые плагины и запускает их. Путь к плагину должен быть определен в файле

```
C:\unitess\plugins\pluginslist.csv
```

В этом файле хранится список всех установленных плагинов.
При окончании работы скрипта APM только разрывает сетевое подключение к сервису управления плагином, но не закрывает его! Вы можете закрыть плагин после измерений специальными командами, например

```
PluginSet  ServiceName stop=true event
```

Ваша откомпилированная VI может храниться в любом месте на диске, но лучше разместить ее в папке, используемой по умолчанию:

```
C:\unitess\plugins
```

Чтобы ваш плагин можно было открыть из меню “Плагины” в UniTesS APM и плагин автоматически запускался при старте скрипта, необходимо добавить его описание в файл:

```
C:\unitess\plugins\pluginslist.csv
```

10. Работа с командами ОС

Команда **SystemExe** обеспечивает доступ к управлению командной строкой операционной системы Windows.

Команда **OpenFile** открывает любой указанный файл, с использованием настроек по умолчанию стандартным приложением ОС Windows.

Пример.

```
SystemExe      notepad c:\UniTesS\Example\Scripts\math.uts
SystemExe      cmd  /c start "C:\UniTesS\Example\Scripts" "math.uts"
dir="C:\UniTesS\Example\Scripts"
OpenFile        C:\UniTesS\Example\Scripts\math.uts
```

Для команды **SystemExe** доступны ключи:
dir="path" - устанавливает рабочий каталог,
wait - ожидает выполнения команды,
minimized - минимизирует окно выполняемой программы .

11. Работа с файлами Excel

Скрипт может загружать и обрабатывать информацию из внешних файлов в формате Excel.

Внимание! Возможна работа только с одним файлом.

Формат команды:

```
Excel <имя файла> действие
```

имя файла - наименование файла в формате Excel с расширением *.xlsx
действие - операция, выполняемая над файлом.

Варианты действий:

Load - загрузка файла без диалоговых окон. Если файл не найден, выводится ошибка. По умолчанию поиск выполняется в папке c:\unitess\temp,
LoadDialog - загрузка файла с подтверждением выбора в диалоговом окне. Если задан путь к папке без указания файла, то выводится диалог с папкой по умолчанию,
Close - в конце работы обязательно необходимо закрывать сессию.

Доступны ключи:

- show - открывает файл Excel,
- title="текст сообщения" - выводит сообщение для пользователя,
- hide - спрятать excel,
- sheet= - выбор текущего листа (по номеру или названию),
- GetCell(num1,num2) - загрузка значения ячейки (num1 - строка, num2 - колонка),
- GetRow(num1,num2,num3) - загрузка строки num1, начиная с колонки num2 (опционально), количество значений num3 (опционально),
- GetCol (num1,num2,num3) - загрузка колонки num1, начиная со строки num2 (опционально), количество значений num3 (опционально),
- UserSelected - пользователь сам выделяет колонку или строку для импорта. Пользователь должен мышкой выделить необходимые элементы - но только строку или колонку. Двумерные массивы не допускаются.

Пример.

```
Excel Load excel.xlsx
Excel LoadDialog "excel.xlsx" title="Выбор файла коррекции" show
Excel Sheet = 2 # изменение текущего листа, можно указывать номер
Excel Sheet = Лист1       # или название
Excel mem_1=GetCell(1;1)  # загрузка значения из ячейки
```

```
Excel mem_1=GetRow(1)      # загрузка всей первой строки
Excel mem_1=GetRow(1; 5)   # загрузка строки начиная с пятой колонки
Excel mem_1=GetRow(1; 5; 10) # загрузка 10 значений первой строки
начиная с пятой колонки
Excel mem_1=GetCol(1)      # загрузка всей первой колонки
Excel mem_1=UserSelected title="Выбор колонки со значениями" column
checksyntax
```

UserSelected можно использовать с ключами:

- title="текст" - сообщение для пользователя,
- checksyntax - выбираются только цифры, проверка на корректность,
- row - пользователь должен выбрать только строку,
- column - пользователь должен выбрать только колонку.

Для close доступны ключи:

- save - сохранение изменений в файле без запросов,
- SaveDialog - вывод диалогового окна: сохранить файл или нет.

Приложение 1. Математические функции

Перечень математических функций.

Функция	Название	Описание
abs(*x*)	Абсолютная величина	Возвращает значение по модулю от величины x.
acos(*x*)	Арккосинус	Возвращает арккосинус числа x.
acosh(*x*)	Гиперболический арккосинус	Возвращает гиперболический арккосинус числа x. Число должно быть больше или равно 1.
asin(*x*)	Арксинус	Возвращает арксинус числа x.
asinh(*x*)	Гиперболический арксинус	Возвращает гиперболический арксинус числа x.
atan(*x*)	Арктангенс	Возвращает арктангенс числа x.
atanh(*x*)	Гиперболический арктангенс	Возвращает гиперболический арктангенс числа x.
ceil(*x*)	Округление в большую сторону	Округляет x до ближайшего целого значения в большую сторону.
ci(*x*)	Интегральный косинус	Возвращает Интегральный косинус числа *x*.
cos(*x*)	Косинус	Возвращает косинус x.
cosh(*x*)	Гиперболический косинус	Возвращает гиперболический косинус числа x.
cot(*x*)	Котангенс	Возвращает котангенс числа x.
csc(*x*)	Косеканс	Возвращает косеканс числа x.
exp(*x*)	Экспонента	Возвращает значение числа e, возведенное в степень x.
expm1(*x*)	Экспонента (x) – 1	Возвращает число, на единицу меньшее числа e, возведенного в степень x: ((e^x) – 1).
floor(*x*)	Округление в меньшую сторону	Округляет x до ближайшего меньшего целого.
gamma(*x*)	Гамма-функция	Вычисляет гамма-функцию числа *x*.
getexp(*x*)	мантисса и экспонента	Возвращает экспоненту числа x.
getman(*x*)	мантисса и экспонента	Возвращает мантиссу числа x.
int(*x*)	Округление до ближайшего целого	Округляет x до ближайшего целого.
intrz(*x*)	Округление до ближайшего целого	Округляет x до ближайшего целого между x и нулем.

ln(*x*)	Натуральный логарифм	Возвращает натуральный логарифм x (по основанию e).
lnp1(*x*)	Натуральный логарифм (x +1)	Возвращает натуральный логарифм от (x + 1).
log(*x*)	Десятичный логарифм	Возвращает десятичный логарифм от x (по основанию 10).
log2(*x*)	Логарифм по основанию 2	Возвращает логарифм от x по основанию 2.
rand( )	Случайное число (0 – 1)	Генерирует случайной число в диапазоне (0 ; 1).
sec(*x*)	Секанс	Возвращает секанс числа x.
si(*x*)	Интегральный синус	Возвращает интегральный синус числа *x*.
sign(*x*)	Кусочно-постоянная функция	Возвращает: 1, если x больше 0 0, если x равен нулю –1, если x меньше 0.
sin(*x*)	Синус	Возвращает синус числа x.
sinc(*x*)	Кардинальный синус	Возвращает кардинальный синус числа x.
sinh(*x*)	Гиперболический синус	Возвращает гиперболический синус x.
spike(*x*)	Импульс	Генерирует спайк-функцию числа *x*.
sqrt(*x*)	Квадратный корень	Возвращает квадратный корень числа x.
step(*x*)	Ступенька	Генерирует степ-функцию для числа *x*.
tan(*x*)	Тангенс	Возвращает тангенс числа x.
tanh(*x*)	Гиперболический тангенс	Возвращает гиперболический тангенс числа x.

Для тригонометрических функций аргументом является значение в радианах.

Приложение 2. Математические операции и работа с массивами

Перечень математических операций

^	Возведение в степень (Пример: math mem_1 = mem_2^2).
+, -, !, ~, ++, – –	Унарные операции сложения, вычитания, логического отрицания, побитового дополнения, пре- и пост инкремента и декремента.
*, /, %	Умножение, деление, целая часть и остаток от деления.

Описание функций работы с массивами

В скрипте, помимо математических, доступны дополнительные функции работы с массивами:

++	Добавление элемента в массив.
Get	Извлечь элемент из массива.
Max	Возвращает элемент массива с наибольшим значением.
Mean	Вычисляет среднее арифметическое значение всех элементов массива.
Median	Вычисляет медианное значение всех элементов массива.
Min	Возвращает элемент массива с наименьшим значением.
Range	Вычисляет разность между максимальным и минимальным значениями из массива.
Rms	Вычисляет среднее квадратическое значение элементов массива.
Size	Возвращает размер массива.
Stdev	Вычисляет стандартное отклонение элементов массива.
Variance	Вычисляет дисперсию элементов массива.
SortUp	Сортировка снизу-вверх
SortDown	Сортировка сверху-вниз

Приложение 3. Форма записи математических операций

Форма записи математических операций для команды Math

Форма записи	**Описание**	**Пример**
++mem_X = mem_Y	*Добавить элемент mem_Y к массиву mem_X*	math ++mem_2 = mem_1
mem_X= get(mem_Y;n)	*Извлечь значение элемента номер n из массива mem_Y и записать в ячейку памяти mem_X*	math mem_3 = get (mem_2; 5) math mem_3 = mem_2[5]
mem_X= max(mem_Y)	*Найти максимальное значение элемента массива mem_Y, и записать в ячейку памяти mem_X*	math mem_2= max(mem_3)
mem_X= mean(mem_Y)	*Записать в ячейку памяти mem_X, среднее арифметическое значение элементов массива mem_Y*	math mem_3= mean(mem_2)
mem_X= median(mem_Y)	*Найти медианное значение элементов массива mem_Y, и записать в ячейку памяти mem_X*	math mem_1= median(mem_4)
mem_X= min(mem_Y)	*Найти минимальное значение элемента массива mem_Y, и записать в ячейку памяти mem_X*	math mem_1= min(mem_4)

mem_X= range(mem_Y)	*Вычислить разность между максимальным и минимальным значениями элементов массива mem_Y, и записать в ячейку памяти mem_X*	math mem_1= range(mem_4)
mem_X= rms(mem_Y)	*Найти среднеквадратическое значение элементов массива mem_Y, и записать в ячейку памяти mem_X*	math mem_1= rms(mem_4)
mem_X= size(mem_Y)	*Вычислить размер массива mem_Y, и записать в ячейку памяти mem_X*	math mem_1= size(mem_4)
mem_x= stdev(mem_Y)	*Вычислить стандартное отклонение элементов массива mem_Y, и записать в ячейку памяти mem_X*	math mem_1= stdev(mem_4)
mem_x= variance(mem_Y)	*Вычислить дисперсию элементов массива mem_Y, и записать в ячейку памяти mem_X*	math mem_1= variance(mem_4)

Приложение 4. Элементы формата

Элементы формата

Таблица описывает синтаксические элементы строки формата. Более детальная информация приведена в таблице примеров формата Приложения №5.

Элемент формата	Описание
%	Начало строки формата.
$ (опционально)	Определяет порядок отображения переменных. Сразу за этим модификатором должно следовать число, которое определяет порядковый номер переменной.
- (опционально)	Выравнивает параметр по левому краю.
+ (опционально)	Устанавливает знак числа слева, даже если число положительное.
w (опционально)	Постфикс отображается через пробел
^ (опционально)	При использовании в строке форматирования кодов преобразования **e** или **g**, этот элемент форматирует число в инженерном виде, где экспонента всегда будет кратна трем.
# (опционально)	Удаляет из числа младшие нулевые разряды.
0 (опционально)	Используется без модификатора «-», дополняет число слева нулями до указанной длины поля. Если элемент отсутствует – число дополняется пробелами.

.Точность или *_ЗначащиеРазряды* (опционально)	Эти модификаторы определяют количество символов для отображения. При использовании модификатора точка, длина поля, указанная за ним, интерпретируется как спецификатор точности для дробной части. Если используется нижнее подчеркивание, число следующее за ним означает число значащих разрядов. *.Точность* — при использовании в записи с плавающей точкой, этот элемент определяет число разрядов справа от десятичной точки. Если параметр опущен, по умолчанию используется шесть разрядов. Если точность указана 0, дробная часть будет опущена. *_ЗначащиеРазряды* — отображает данные, округленные до числа разрядов, указанных пользователем. *.Точность* влияет только на разряды справа от десятичной точки, *_ЗначащиеРазряды* включает в себя все разряды кроме нулей и пробелов. Например ● 3.457 - 4 значащих разряда ● 0.0012 - 2 значащих разряда ● 123000 - 3 значащих разряда **Замечание.** Нельзя использовать одновременно оба этих модификатора в одной строке формата.
Коды преобразования	Символы, определяющие порядок форматирования параметра.
	Коды преобразования для целочисленных типов и чисел с фиксированной точкой: ● x — Шестнадцатеричное целочисленное (например, B8). ● o — Восьмеричное целочисленное (например, 701). ● b — Двоичное целочисленное (например, 1011). ● d — Десятичное целочисленное со знаком. ● u — Десятичное целочисленное без знака.

	Коды преобразования для чисел с плавающей и фиксированной точкой: ● f — Число с плавающей точкой с обычным форматом дробной части (например, 12.345). ● e — Число с плавающей точкой в экспоненциальной записи (например, 1.234E1). ● g — Программа использует формат f или e в зависимости от показателя степени числа. Код f будет использован если показатель степени больше –4 или меньше, чем указанная точность. Код e будет использован если показатель степени меньше –4 или больше, чем указанная точность. ● p — Число с плавающей точкой в записи с постфиксом в соответствии с системой СИ.	
	Коды преобразования для строк: ● s — Строка (например, abc).	
Коды десятичного знака	Определяет какой знак разделяет целую и дробную части числа. Этот код действует до следующего места, где встречается аналогичный код, изменяющий действие текущего.	
	%,;	Десятичный знак - запятая.
	%.;	Десятичный знак - точка.
	%;	Десятичный знак – используемый системой по умолчанию.

Приложение 5. Примеры описания форматов

Таблица содержит примеры использования синтаксических элементов в строке спецификатора формата.

Тип	Аргумент	Строка формата	Результирующая строка	Пояснения
Автоматическое форматирование (%g)	12.00	%#g	12	Спецификатор # : LabVIEW убирает конечные нули. Спецификатор g : LabVIEW выбирает экспоненциальное представление либо представление с плавающей точкой, в зависимости от форматируемого числа.
	12000000	%#g	1.2E+6	
Десятичное (%d)	12.67	Выполнено= %d%%	Выполнено = 13%	Спецификатор %d : LabVIEW округляет аргумент. Используйте %% чтобы получить символ % в результирующей строке.
Представление с плавающей точкой (%f)	12.67	Температура: %5.1f	Температура: 12.7	Число 5 в строке формата определяет ширину поля , число 1 – число разрядов справа от десятичной точки или точность.
Экспоненциальное представление (%e)	12.67	%.3e	1.267E+1	Добавьте к записи символ ^, чтобы перейти к инженерной записи, где экспонента всегда кратна трем.
	12.67	%^.3e	12.670E+0	
Запись в системе СИ (%p) или (%r)	12000000	%.2p	12.00M	Запись .2 определяет, точность в два разряда в результирующей строке. Запись _2 определяет два значащих разряда в результирующей строке. %p добавляет постфиксы в соответствии с системой СИ. %r добавляет русские постфиксы в системе СИ
	12000000	%_2p	12M	
	0,00006	%_2r	60мк	
Шестнадцатеричное представление (%x)	12	%02x	0C	Знак – выравнивает запись по левому краю, а 0 (ноль) дополняет число с левой

Восьмеричное (%o)	12	%06o	000014	стороны нулями. Число обозначает желаемую ширину результирующей строки. Другими словами запись –n дополняет результирующую строку пробелами до ширины *n*. b обозначает основание 2, o обозначает основание 8, x обозначает основание 16.
Двоичное (%b)	12	%b	1100	
Строка (%s)	Иванов Иван	Имя: %s, %s.	Имя: Иванов, Иван.	Во втором примере формат определяет использование максимум шести символов из строки Hello, World, затем дополнить пробелами, чтобы общая длина строки составила 10.
	Hello, World	Строка: %10.6s	Строка: Hello,	
Варьируемый порядок (%$)	Иванов Иван	%2$s %1$s	Иван Иванов	В данном примере Иванов – первая переменная, Иван – вторая переменная. При использовании синтаксического элемента $ в строке формата для одной из переменных, последующие элементы, которым не предшествует символ $, относятся к следующей переменной. При использовании синтаксического элемента $ для одной из переменных в строке, используйте его и для других для улучшения читаемости.

Приложение 6. Scope - класс осциллографов

Перечень команд, доступных для IVI класса осциллографов с допустимыми параметрами.

IVIDriverSet – отправка команд		
№	**Команда**	**Описание**
1	CHANNEL	Номер канала
2	TIMEOUT	Таймаут
3	ACQTYPE	Тип захвата
4	HORIZONTALRANGE	Коэффициент развертки
5	CHANNELENABLED	Включение канала
6	PROBEATTENUATION	Коэффициент масштабирования
7	VERTICALRANGE	Коэффициент отклонения
8	VERTICALOFFSET	Вертикальное смещение
9	VERTICALCOUPLING	Связь по входу
10	MAXIMUMINPUTFREQUENCY	Максимальная частота входного сигнала
11	INPUTIMPEDANCE	Входное сопротивление
12	TRIGGERTYPE	Тип синхронизации
13	TRIGGERSOURCE	Источник синхронизации
14	TRIGGERCOUPLING	Связь по входу синхронизации
15	TRIGGERHOLDOFF	Задержка синхронизации
16	TRIGGERLEVEL	Уровень синхронизации
17	TRIGGERSLOPE	Фронт импульса синхронизации
18	TRIGGERMODE	Режим синхронизации
19	HIGHREF	Верхний опорный уровень
20	LOWREF	Нижний опорный уровень
21	MIDREF	Средний опорный уровень

22	RESET	Сброс
23	AUTOSETUP	Автонастройка
24	VISA	Отправка команды прибору через VISA интерфейс
25	RUNCONTROL	Тип измерения (однократно, постоянно)
26	ACQINIT	Запуск развертки
27	WAIT	Ожидание выполнения предыдущих действий

IVIDriverGet – Захват данных		
№	**Команда**	**Описание**
1	SELFTEST	Самодиагностика
2	VISA	Отправка команды на запрос данных через VISA интерфейс
3	RISETIME	Длительность фронта нарастания импульса
4	FALLTIME	Длительность фронта спада импульса
5	FREQUENCY	Частота
6	PERIOD	Период
7	VOLTAGERMS	Среднеквадратическое значение
8	VOLTAGERMSCYCLE	Среднеквадратическое значение за период
9	VOLTAGEMIN	Минимальное значение
10	VOLTAGEMAX	Максимальное значение
11	VOLTAGEVPP	Размах
12	VOLTAGELOW	Нижнее значение
13	VOLTAGEHIGH	Верхнее значение
14	VOLTAGEAVERAGE	Среднее арифметическое значение
15	VOLTAGEAVERAGECYCLE	Среднее арифметическое значение за период
16	WIDTHNEGATIVE	Ширина отрицательного импульса
17	WIDTHPOSITIVE	Ширина положительного импульса
18	DUTYCYCLENEGATIVE	Отрицательный рабочий цикл сигнала

19	DUTYCYCLEPOSITIVE	Положительный рабочий цикл сигнала
20	AMPLITUDE	Амплитуда сигнала
21	OVERSHOOT	Выброс
22	PRESHOOT	Провал

Подробное описание и примеры команд для класса осциллографов.

IVIDriverSet – отправка команд.			
Команда	**Описание**	**Аргумент**	**Пример**
CHANNEL	Выбор номера канала. Последующие установки будут применяться для выбранного канала.	<целочисленное значение> x >= 1	IVIdriverSet scope channel=1
TIMEOUT	Максимальное время ожидания ответа от прибора, мс.	<целочисленное значение> 1 <= x <=6000	IVIdriverSet scope timeout=5000
ACQTYPE	Тип захвата сигнала	NORMAL [NORM]	IVIdriverSet scope ACQTYPE=NORM
		PEAK	
		HIGH_RESOLUTION [HIGHRESOLUTION, HIRES]	
		ENVELOPE [ENV]	
		AVERAGE [AVG]	
HORIZONTALRANGE (HR, HRANGE)	Коэффициент развертки	< численное значение >	IVIdriverSet scope HRANGE=5m
ACQUISITIONSTARTTIME (ACQSTARTTIME, ACQSTART)	Время начала захвата данных	< численное значение >	IVIdriverSet scope ACQSTARTTIME=-2m
CHANNELENABLED (CHEN)	Включение канала	On	IVIdriverSet scope CHANNELENABLED=On
		Off	
		Enabled	
		Disabled	

		En	
		Dis	
		1	
		0	
PROBEATTENUATION (PROBEATT)	Коэффициент масштабирования для разных щупов	< численное значение >	IVIdriverSet scope PROBEATT=10
VERTICALRANGE (VRANGE, VR)	Коэффициент отклонения	< численное значение >	IVIdriverSet scope VRANGE=500m
VERTICALOFFSET (VOFFSET, VO)	Вертикальное смещение	< численное значение >	IVIdriverSet scope VOFFSET=1
VERTICALCOUPLING (VCOUPLING, COUPLING)	Связь входного сигнала для канала	AC	IVIdriverSet scope VCOUPLING=AC
		DC	
		GROUND	
		GND	
MAXIMUMINPUTFREQUENCY (MAXINPUTFREQ)	Максимальная входная частота	< численное значение >	IVIdriverSet scope MAXINPUTFREQ=200M
INPUTIMPEDANCE (INPUTZ, INZ)	входное сопротивление канала, Ом	50	IVIdriverSet scope INPUTZ=50
		1M	
TRIGGERTYPE (TRIGTYPE)	Определяет событие, которое запускает развертку	EDGE	IVIdriverSet scope TRIGTYPE=EDGE
		RUNT	
		GLITCH	
		WIDTH	
		TV	
		AC_LINE	
		IMMEDIATE	
TRIGGERSOURCE (TRIGSOURCE, TRIGSRC)	источник синхронизации	<номер канала>	IVIdriverSet scope TRIGSOURCE=1
		EXTERNAL [EXT]	
TRIGGERCOUPLING (TRIGCOUPLING, TRIGCPLG)	Тип связи синхронизации. Выбирается в	AC	IVIdriverSet scope TRIGCOUPLING=AC
		DC	
		LF_REJECT	
		HF_REJECT	

	зависимости от вида сигнала синхронизации.	NOISE_REJECT	
TRIGGERHOLDOFF (TRIGHOLDOFF, TRIGHLDOFF)	Задержка синхронизации	< численное значение >	IVIdriverSet scope TRIGHOLDOFF=5m
TRIGGERLEVEL (TRIGLEVEL)	Уровень синхронизации	< численное значение >	IVIdriverSet scope TRIGLEVEL=1.5
TRIGGERSLOPE (TRIGSLOPE)	Фронт импульса синхронизации	POSITIVE (POS)	IVIdriverSet scope TRIGSLOPE=POS
		NEGATIVE (NEG)	
TRIGGERMODE (TRIGMODE, TM)	Режим синхронизации	MANUAL	IVIdriverSet scope TRIGMODE=MAN
		MAN	
		AUTO_LEVEL	
		AUTO	
HIGHREF	Верхний опорный уровень	< численное значение >	IVIdriverSet scope HIGHREF=5
LOWREF	Нижний опорный уровень	< численное значение >	IVIdriverSet scope LOWREF=-5
MIDREF	Средний опорный уровень	< численное значение >	IVIdriverSet scope MIDREF=2.5
RESET	Сбрасывает текущие настройки на настройки по умолчанию		IVIdriverSet scope RESET
AUTOSETUP	Автоматическая настройка		IVIdriverSet scope AUTOSETUP
VISA	Отправка команды в прибор через VISA интерфейс	<строка команды>	IVIdriverSet scope VISA=*IDN?
RUNCONTROL (RUNCTR)	Режим измерения: однократный или непрерывный	CONTINIOUS(CONT)	IVIdriverSet scope RUNCTR=CONT
		SINGLE(RUN)	
ACQUISITIONINIT (ACQINIT, ACQON, ACQUISITIONON)	Запуск развертки		IVIdriverSet scope ACQINIT

WAIT (DONE?, WAITDONE, WAITUNTILDONE)	Ожидание завершения предыдущих операций		IVIdriverSet scope WAITDONE

Приложение 7. DMM - цифровые мультиметры

Перечень команд, доступных для IVI класса цифровых мультиметров с допустимыми параметрами.

IVIDriverSet – отправка команд		
№	**Команда**	**Описание**
1	VISA	Отправка команды прибору через VISA интерфейс
2	TIMEOUT	Таймаут
3	RESET	Сброс всех параметров и перезагрузка
4	AUTOZERO	Установка нуля
5	POWERLINEFREQ	Частота питающей сети
6	CHANNEL	Текущий канал
7	CHANNELENABLED	Включение канала
8	FUNCTION	Измеряемый параметр
9	RANGE	Диапазон измерения
10	ACQUISITIONINIT	Инициирует измерение с текущими настройками
11	TRIGGERSOURCE	Источник синхронизации
12	TRIGGERSLOPE	Фронт синхронизации
13	TRIGGERDELAY	Задержка синхронизации

IVIDriverGet – Захват данных		
№	**Команда**	**Описание**
1	SELFTEST	Самодиагностика
2	FETCH	Значение последнего измерения
3	READ	Текущее измерение
4	VISA	Отправка команды через VISA интерфейс

Подробное описание и примеры команд для цифровых мультиметров.

IVIDriverSet – отправка команд.			
Команда	**Описание**	**Аргумент**	**Пример**
VISA	Отправка команды в прибор через VISA интерфейс	<команда>	IVIDriverSet mmeter VISA=*IDN?
TIMEOUT	Максимальное время ожидания ответа от прибора, мс.	< численное значение, мс>	IVIDriverSet mmeter TIMEOUT=5000
RESET	Сброс всех параметров и перезагрузка		IVIDriverSet mmeter RESET
AUTOZERO	Установка нуля. Текущее значение принимается за ноль и вычитается из последующих измеренных значений.		IVIDriverSet mmeter AUTOZERO
POWERLINEFREQ	Частота питающей сети	<численное значение, Гц>	IVIDriverSet mmeter POWERLINEFREQ=50
CHANNEL	Установка текущего канала	<целочисленное значение >	IVIDriverSet mmeter CHANNEL=1
CHANNELENABLED (CHANNELENABLE, CHEN)	Включение/выключение текущего канала	On	IVIDriverSet mmeter CHEN=On
		Off	
		Enable	
		Enabled	
		Disable	
		Disabled	
		En	
		Dis	
		1	
		0	

FUNCTION (FUNC)	Измеряемый параметр	VDC	IVIDriverSet mmeter FUNC=VDC
		VAC	
		IDC	
		IAC	
		R	
		R2	
		R2WIRE	
		R4	
		R4WIRE	
		V_AC_DC	
		I_AC_DC	
		F	
		P	
		T	
RANGE	Диапазон измерений	< численное значение >	IVIDriverSet mmeter RANGE=700
ACQUISITIONINIT (ACQINIT, ACQON, ACQUISITIONON)	Запуск измерения с текущими настройками. Если какие-то параметры не заданы, берутся настройки по умолчанию.		IVIDriverSet mmeter ACQINIT
TRIGGERSOURCE (TRIGGER, TRIGGERTYPE)	Выбор источника синхронизации	SOFT	IVIDriverSet mmeter TRIGGERSOURCE=EXT
		SOFTWARE	
		EXT	
		EXTERNAL	
		IMM	
		IMMEDIATE	
TRIGGERSLOPE (TRIGSLOPE)	Выбор фронта синхронизации	POS	IVIDriverSet mmeter TRIGSLOPE=POS
		POSITIVE	

		NEG	
		NEGATIVE	
TRIGGERDELAY (TRIGDELAY)	Установка задержки синхронизации	<численное значение, с>	IVIDriverSet mmeter TRIGDELAY=500m

IVIDriverGet – Захват данных.

Команда	Описание	Возвращаемое значение	Пример
SELFTEST	Самодиагностика прибора	[0,1] 0 – pass, 1 – fail.	IVIdriverGet mmeter mem_1= SELFTEST
FETCH	Считывание значения последнего измерения	< численное значение >	IVIdriverGet mmeter mem_1= FETCH
READ	Запуск измерения и считывание нового значения	< численное значение >	IVIdriverGet mmeter mem_1= READ
VISA	Отправка команды на запрос данных через VISA интерфейс	< численное значение >	IVIdriverGet mmeter mem_1= VISA(:MEAS:VAMP?)

Приложение 8. СВЧ генераторы

Перечень команд, доступных для IVI класса RF Sig Gen СВЧ генераторов с допустимыми параметрами.

IVIDriverSet – отправка команд		
№	**Команда**	**Описание**
1	VISA	Отправка команды прибору через VISA интерфейс
2	TIMEOUT	Таймаут
3	RESET	Сброс всех параметров и перезагрузка
4	CHANNEL	Выбор текущего канала
5	WAITUNTILDONE	Ожидать выполнения очереди команд
6	FREQUENCY	Задание частоты генератора
7	POWERLEVEL	Уровень выходной мощности
8	CHANNELENABLED	Включение канала

IVIDriverGet – Захват данных		
№	**Команда**	**Описание**
1	SELFTEST	Самодиагностика
2	VISA	Отправка команды через VISA интерфейс

Подробное описание команд и примеры для класса СВЧ генераторов.

IVIDriverSet – отправка команд.			
Команда	**Описание**	**Аргумент**	**Пример**
VISA	Отправка команды в прибор через VISA интерфейс	<команда>	IVIDriverSet generator VISA=*IDN?
TIMEOUT	Максимальное время ожидания ответа от прибора, мс.	< численное значение, мс>	IVIDriverSet generator TIMEOUT=5000

RESET	Сброс всех параметров и перезагрузка		IVIDriverSet generator RESET
CHANNEL	Выбор текущего канала	<целочисленное значение>	IVIDriverSet generator CHANNEL=1
WAITUNTILDONE (DONE, WAIT, WAITDONE)	Ожидание выполнения очереди команд		IVIDriverSet generator WAITUNTILDONE
FREQUENCY (FREQ, F)	Задание частоты генератора	<численное значение,Гц>	IVIDriverSet generator FREQ=1,8G
POWERLEVEL (LEVEL, POWER)	Уровень выходной мощности	<численное значение, дБм>	IVIDriverSet generator POWER=10
CHANNELENABLE (CHANNELENABLED, CHEN)	Включение/выключение текущего канала	On	IVIDriverSet generator CHEN=1
		Off	
		Enable	
		Enabled	
		Disable	
		Disabled	
		En	
		Dis	
		1	
		0	

IVIDriverGet – Захват данных.

Команда	Описание	Возвращаемое значение	Пример
SELFTEST	Самодиагностика прибора	[0,1] 0 – pass, 1 – fail.	IVIdriverGet generator mem_1= SELFTEST

VISA	Отправка команды на запрос данных через VISA интерфейс	< численное значение >	IVIdriverGet generator mem_1= VISA(:MEAS:POUT?)

Приложение 9. Анализаторы спектра

Перечень команд, доступных для IVI класса Spec An - анализаторов спектра с допустимыми параметрами.

IVIDriverSet – отправка команд.		
№	**Команда**	**Описание**
1	VISA	Отправка команды прибору через VISA интерфейс
2	TIMEOUT	Таймаут
3	RESET	Сброс всех параметров и перезагрузка
4	ACQUISITIONINIT	Начинает захват сигнала
5	WAITUNTILDONE	Ожидание завершения захвата данных
6	CENTRALFREQUENCY	Центральная частота
7	STARTFREQUENCY	Начальная частота
8	STOPFREQUENCY	Конечная частота
9	FREQUENCYOFFSET	Смещение частоты
10	SPAN	Занимаемая полоса
11	SWEEPMODE	Метод свипирования
12	SWEEPTIME	Скорость свипирования
13	REFERENCELEVEL	Опорный уровень
14	REFLEVELOFFSET	Смещение опорного уровня
15	ATTENUATION	Автоматический выбор ослабления
16	INPUTIMPEDANCE	Входное сопротивление
17	UNIT	Единицы амплитуды
18	VERTSCALE	Тип шкалы по вертикали
19	TRACE	Выбор текущего трэйса
20	TRACEMODE	Тип трэйса
21	NUMBEROFSWEEPS	Количество проходов свипа

22	VBW	Полоса видеофильтра
23	RBW	Полоса радиофильтра
24	DETECTOR	Тип детектора
25	TRIGGER	Источник синхронизации
26	EXTERNALTRIGGERSLOPE	Фронт внешнего сигнала синхронизации
27	EXTERNALTRIGGERLEVEL	Уровень внешнего сигнала синхронизации
28	VIDEOTRIGGERLEVEL	Уровень видеосигнала для запуска синхронизации
29	VIDEOTRIGGERSLOPE	Фронт видеосигнала для запуска синхронизации
30	MARKER	Текущий маркер
31	MARKERENABLE	Включение маркера
32	MARKERDELTAEN	Делает текущий маркер активным для операций с маркерами
33	SETMARKERTRACE	Назначение трэйса текущему маркеру
34	MARKERSEARCH	Поиск маркера
35	MARKERMOVELEFT	Сдвиг маркера влево
36	MARKERFREQCOUNTENABLE	Включение частотомера для маркера
37	MARKERFREQCOUNTRES	Разрешение частотомера маркера, Гц
38	MARKERFREQUENCY	Положение текущего маркера
39	MARKERDISABLEALL	Выключение всех маркеров
40	MARKERSETCF	Установка центральной частоты в соответствии со значением активного маркера
41	MARKERSETREFLEV	Установка опорного уровня в соответствии со значением активного маркера

IVIDriverGet – Захват данных		
№	**Команда**	**Описание**
1	SELFTEST	Самодиагностика
2	VISA	Отправка команды через VISA интерфейс
3	MARKERFREQ	Значение частоты маркера

4	MARKERLEVEL	Значение уровня маркера
5	REFMARKERLEVEL	Значение амплитуды опорного маркера
6	REFMARKERFREQ	Значение частоты опорного маркера
7	SWEEPTIME	Значение времени свипа

Подробное описание и примеры команд для анализаторов спектра.

IVIDriverSet – отправка команд			
Команда	**Описание**	**Аргумент**	**Пример**
VISA	Отправка команды в прибор через VISA интерфейс	<команда>	IVIDriverSet specan VISA=*IDN?
TIMEOUT	Максимальное время ожидания ответа от прибора, мс.	<численное значение, мс>	IVIDriverSet specan TIMEOUT=5000
RESET	Сброс всех параметров и перезагрузка		IVIDriverSet specan RESET
ACQUISITIONINIT (ACQINIT,ACQON, ACQUISITIONON)	Запуск захвата сигнала		IVIDriverSet specan ACQINIT
WAITUNTILDONE (DONE,WAIT, WAITDONE)	Ожидание завершения захвата данных		IVIDriverSet specan WAITDONE
CENTRALFREQUENCY(CF, CFREQ)	Установка центральной частоты	<численное значение, Гц>	IVIDriverSet specan CFREQ=1800M
STARTFREQUENCY (STARTFREQ)	Установка начальной частоты	<численное значение, Гц>	IVIDriverSet specan STARTFREQ=1750 M
STOPFREQUENCY (STOPFREQ)	Установка конечной частоты	<численное значение, Гц>	IVIDriverSet specan STOPFREQ=1850M
FREQUENCYOFFSET (FO, FOFFSET, FREQOFFSET)	Установка значения смещения частоты, которое прибавляется к считываемым показаниям	<численное значение, Гц>	IVIDriverSet specan FREQOFFSET=10 M

SPAN	Диапазон частот анализируемого спектра (от начальной частоты до конечной)	<численное значение, Гц>	IVIDriverSet specan SPAN=100M
SWEEPMODE	Режим свипирования (Постоянный или однократный)	CONT	IVIDriverSet specan SWEEPMODE=CONT
		CONTINUOUS	
		SINGLE	
SWEEPTIME	Время свипирования	<численное значение, с>	IVIDriverSet specan SWEEPTIME=500m
REFERENCELEVEL (REFLEVEL, RL)	Установка опорного уровня	<численное значение, UNIT>	IVIDriverSet specan REFLEVEL=30
REFLEVELOFFSET (RLOFFSET)	Установка предзатухания или предусиления сигнала	<численное значение, UNIT>	IVIDriverSet specan RLOFFSET=1,5
ATTENUATION (ATT)	Включение функции автоматического ослабления	AUTO	IVIDriverSet specan ATTENUATION=-10
		<численное значение, UNIT >	
INPUTIMPEDANCE (IMPEDANCE)	Входное сопротивление	<численное значение, Ом>	IVIDriverSet specan IMPEDANCE=75
UNIT	Выбор единиц для входного, выходного и отображаемого значения амплитуды	DBM	IVIDriverSet specan UNIT=dBm
		DBMV	
		DBUV	
		V	
		VOLT	
		W	
		WATT	
VERTSCALE (SCALE)	Выбор типа шкалы по вертикали(логарифмическая либо линейная)	LIN	IVIDriverSet specan VERTSCALE=Lin
		LOG	
TRACE	Установка текущего трэйса	<целочисленное значение>	IVIDriverSet specan TRACE=1
TRACEMODE (TRACETYPE)	Устанавливает режим сбора данных	CLEAR	IVIDriverSet specan TRACEMODE=AVE
		MAXHOLD	
		MINHOLD	

		AVE	
		AVERAGE	
		VIEW	
		STORE	
NUMBEROFSWEEPS (NUMSWEEPS, SWEEPS)	Количество проходов сбора данных	<целочисленное значение>	IVIDriverSet specan NUMSWEEPS=16
VBW	Полоса видеофильтра	AUTO	IVIDriverSet specan VBW=5k
		<численное значение, Гц>	
RBW	Полоса радиофильтра	AUTO	IVIDriverSet specan RBW=10k
		<численное значение, Гц>	
DETECTOR (DET)	Тип детектора	P	IVIDriverSet specan DETECTOR=peak
		PEAK	
		PK	
		A	
		AVE	
		AVERAGE	
		MAXP	
		MAXPEAK	
		POSITIVEPEAK	
		POSP	
		POSPEAK	
		MINP	
		MINPEAK	
		NEGATIVEPEAK	
		NEGP	
		NEGPEAK	

		SAMPLE	
		RMS	
TRIGGER	Тип синхронизации	EXT	IVIDriverSet specan TRIGGER=ext
		EXTERNAL	
		IMM	
		IMMEDIATE	
		SOFT	
		SOFTWARE	
		AC	
		VIDEO	
EXTERNALTRIGGERSLOPE (EXTTRIGSLOPE)	Установка фронта синхронизации. Если синхронизация внешняя.	POS	IVIDriverSet specan EXTTRIGSLOPE=pos
		POSITIVE	
		NEG	
		NEGATIVE	
EXTERNALTRIGGERLEVEL (EXTTRIGLEV)	Установка уровня синхронизации. Если синхронизация внешняя.	<численное значение, UNIT >	IVIDriverSet specan EXTTRIGLEV=-30
VIDEOTRIGGERLEVEL (VIDTRIGLEV)	Установка уровня синхронизации по видеосигналу.	<численное значение, UNIT >	IVIDriverSet specan VIDTRIGLEV=-10
VIDEOTRIGGERSLOPE (VIDTRIGSLOPE)	Установка фронта синхронизации по видеосигналу.	POS	IVIDriverSet specan VIDTRIGSLOPE=pos
		POSITIVE	
		NEG	
		NEGATIVE	
MARKER	Установка текущего маркера	<целочисленное значение>	IVIDriverSet specan MARKER=2
MARKERENABLE (MKEN)	Включение текущего маркера	On	IVIDriverSet specan MKEN=On
		Off	
		Enable	
		Enabled	

		Disable	
		Disabled	
		En	
		Dis	
		1	
		0	
MARKERDELTAEN (MKDELTAEN)	Делает текущий маркер активным для операций с маркерами	On	IVIDriverSet specan MKDELTAEN=Off
		Off	
		Enable	
		Enabled	
		Disable	
		Disabled	
		En	
		Dis	
		1	
		0	
SETMARKERTRACE	Назначение трэйса текущему маркеру	<целочисленное значение>	IVIDriverSet specan SETMARKERTRACE=1
MARKERSEARCH (MKSEARCH)	Поиск маркера по параметрам	EAK	IVIDriverSet specan MKSEARCH=Min
		MIN	
		MINIMUM	
		NEXT	
		NEXTPEAK	
		NEXTLEFT	
		NEXTRIGHT	
MARKERMOVELEFT (MKMOVEL, MKMOVELEFT)	Сдвиг маркера влево на минимальный шаг		IVIDriverSet specan MKMOVELEFT

MARKERFREQCOUNTENABLE (MKFREQCOUNTEN)	Включение частотомера для маркера для получения более точного значения.	On	IVIDriverSet specan MKFREQCOUNTEN=En
		Off	
		Enable	
		Enabled	
		Disable	
		Disabled	
		En	
		Dis	
		1	
		0	
MARKERFREQCOUNTRES (MKFREQCOUNTRES)	Разрешение частотомера маркера	<численное значение, Гц>	IVIDriverSet specan MKFREQCOUNTRES=100
MARKERFREQUENCY (MARKERFREQ, MARKERTIME, MKFR, MKTIME)	Установка положения текущего маркера	<численное значение, Гц>	IVIDriverSet specan MARKERFREQ=1790M
MARKERDISABLEALL (MKDISABLEALL)	Выключение всех маркеров		IVIDriverSet specan MKDISABLEALL
MARKERSETCF	Установка центральной частоты в соответствии со значением активного маркера		IVIDriverSet specan MARKERSETCF
MARKERSETREFLEV	Установка опорного уровня в соответствии со значением активного маркера		IVIDriverSet specan MARKERSETREFLEV

IVIDriverGet – Захват данных.

Команда	Описание	Возвращаемое значение	Пример
SELFTEST	Самодиагностика прибора	[0,1] 0 – pass, 1 – fail.	IVIdriverGet specan mem_1= SELFTEST

VISA	Отправка команды на запрос данных через VISA интерфейс	< численное значение >	IVIdriverGet specan mem_1= VISA(:MEAS:POUT?)
MARKERFREQ (MARKERTIME, MKFR, MKTIME)	Значение частоты маркера	<численное значение, Гц>	IVIdriverGet specan mem_1= MKFR
MARKERLEVEL (MKLEV)	Значение амплитуды маркера	<численное значение, UNIT >	IVIdriverGet specan mem_1= MKLEV
REFMARKERLEVEL (RMKLEV)	Значение амплитуды опорного маркера	<численное значение, UNIT >	IVIdriverGet specan mem_1= RMKLEV
REFMARKERFREQ (REFMARKERTIME, RMKFR, RMKTIME)	Значение частоты опорного маркера	<численное значение, Гц>	IVIdriverGet specan mem_1= REFMARKERFREQ
SWEEPTIME	Значение времени свипа	<численное значение, с>	IVIdriverGet specan mem_1= SWEEPTIME

Приложение 10. Осциллограф Fluke 12x

Перечень команд, доступных для осциллографов Fluke серии 12x с допустимыми параметрами.

DriverSet f12x – отправка команд		
№	**Команда**	**Описание**
1	CHANNEL	Выбор текущего канала
2	TIMEOUT	Таймаут
3	CHANNELENABLED	Включение текущего канала
4	RESET	Сброс всех параметров и перезагрузка
5	AUTOSETUP	Автоматическая настройка
6	VERTICALRANGE	Коэффициент отклонения
7	HORIZONTALRANGE	Коэффициент развертки
8	CAPRANGE	Диапазон измерения ёмкости
9	RESRANGE	Диапазон измерения сопротивления
10	COUPLING	Связь входного сигнала
11	PROBEATTENUATION	Коэффициент масштабирования
12	PROBESENSITIVITY	Чувствительность щупа
13	SETMEASUREMENT	Установка измеряемого параметра
14	TRIGGERSLOPE	Фронт синхронизации
15	TRIGGERSOURCE	Источник синхронизации
16	TRIGGERLEVEL	Уровень синхронизации
17	TRIGGERMODE	Режим синхронизации
18	HORIZONTALOFFSET	Смещение по горизонтали
19	VERTICALOFFSET	Смещение по вертикали

DriverGet f12x – Захват данных		
№	**Команда**	**Описание**
1	MAIN_A	Считывание показаний из поля MAIN для канала A
2	MAIN_B	Считывание показаний из поля MAIN для канала B
3	FALLTIME	Время спада импульса
4	RISETIME	Время нарастания импульса
5	PERIOD	Значение периода сигнала

Подробное описание команд и примеры.

DriverSet f12x – отправка команд			
Команда	**Описание**	**Аргумент**	**Пример**
CHANNEL	Выбор текущего канала	A (1)	DriverSet f12x CHANNEL=A
		B (2)	
TIMEOUT	Максимальное время ожидания ответа от прибора	<численное значение, мс>	DriverSet f12x TIMEOUT=5000
CHANNELENABLED (CHEN)	Включение/ выключение текущего канала	On	DriverSet f12x CHEN=On
		Off	
		Enable	
		Enabled	
		Disable	
		Disabled	
		En	
		Dis	
		1	
		0	
RESET	Сброс всех параметров и перезагрузка		DriverSet f12x RESET

AUTOSETUP	Автоматическая настройка. Прибор выбирает оптимальные настройки для стабильного наблюдения текущей формы сигнала		DriverSet f12x AUTOSETUP
VERTICALRANGE (VRANGE, VR)	Коэффициент отклонения.	[5m, 10m, 20m, 50m, 100m, 200m, 500m, 1, 2, 5, 10, 20, 50, 100, 200, 500]	DriverSet f12x VRANGE=10m
HORIZONTALRANGE (HR, HRANGE)	Коэффициент развертки.	[20n, 50n, 100n, 200n, 500n, 1u, 2u, 5u, 10u, 20u, 50u, 100u, 200u, 500u, 1m, 2m, 5m, 10m, 20m, 50m, 100m, 200m, 500m, 1, 2, 5]	DriverSet f12x HRANGE=200n
CAPRANGE	Диапазон измерения ёмкости	[50n, 500n, 5u, 50u, 500u]	DriverSet f12x CAPRANGE=5u
RESRANGE	Диапазон измерения сопротивления	[50, 500, 5k, 50k, 500k, 5M, 30M]	DriverSet f12x RESRANGE=1M
COUPLING	Связь входного сигнала	AC	DriverSet f12x COUPLING=AC
		DC	
PROBEATTENUATION (PROBEATT)	Коэффициент масштабирования	1:1	DriverSet f12x PROBEATT=10:1
		10:1	
		20:1	
		100:1	
		200:1	
		1000:1	
PROBESENSITIVITY (PROBESENS)	Чувствительность щупа	[100uV/A, 1mV/A, 10mV/A, 100mV/A, 400mV/A, 1V/A, 10mV/mA]	DriverSet f12x PROBESENS=1mV/A

SETMEASUREMENT (SETMEAS)	Установка измеряемого параметра. Для второго канала недоступны: RES, CAP, DIODE.	VAC	DriverSet f12x SETMEAS=AAC
		VDC	
		VAC_VDC	
		AAC	
		ADC	
		AAC_ADC	
		DUTYCYCLEPOS	
		DUTYCYCLENEG	
		PULSEWIDTHPOS	
		PULSEWIDTHNEG	
		PEAKMAX	
		PEAK2PEAK (P2P)	
		PEAKMIN	
		TEMP	
		FREQ	
		RES	
		CAP	
		DIODE	
TRIGGERSLOPE (TRIGSLOPE)	Фронт синхронизации	POSITIVE (POS)	DriverSet f12x TRIGSLOPE=POS
		NEGATIVE (NEG)	
TRIGGERSOURCE (TRIGSOURCE, TRIGSRC)	Источник синхронизации	A (1)	DriverSet f12x TRIGSRC=A
		B (2)	
TRIGGERLEVEL (TRIGLEVEL)	Уровень синхронизации	<численное значение, В>	DriverSet f12x TRIGLEVEL=1,5
TRIGGERMODE (TRIGMODE, TM)	Режим синхронизации	FREERUN	DriverSet f12x TRIGMODE=SINGLE
		ONTRIG	
		SINGLE	
		ROLL	

HORIZONTALOFFSET (HOFFSET, HO)	Смещение по горизонтали	<численное значение, с>	DriverSet f12x HOFFSET=25m
VERTICALOFFSET (VOFFSET, VO)	Смещение по вертикали	<численное значение, В>	DriverSet f12x VOFFSET=1

DriverGet f12x – захват данных

Команда	Описание	Возвращаемое значение	Пример
MAIN_A	Считывание показаний из поля MAIN для канала А	<численное значение>	DriverGet f12x mem_1 = Main_A
MAIN_B	Считывание показаний из поля MAIN для канала В	<численное значение>	DriverGet f12x mem_1 = Main_B
FALLTIME	Время спада импульса	<численное значение, с>	DriverGet f12x mem_1 = FALLTIME
RISETIME	Время нарастания импульса	<численное значение, с>	DriverGet f12x mem_1 = RISETIME
PERIOD	Значение периода сигнала	<численное значение, с>	DriverGet f12x mem_1 = PERIOD

Приложение 11. Калибратор-вольтметр В1-28

Перечень команд, доступных для калибратора-вольтметра В1-28 с допустимыми параметрами.

DriverSet B1_28 – отправка команд		
№	**Команда**	**Описание**
1	STOP	Отключение выхода калибратора-вольтметра
2	GO	Подключение выхода калибратора-вольтметра
3	TIMEOUT	Выбор времени ожидания драйвера
4	CONN, ZCOMP	Выбор двух/четырехпроводного режима в режиме воспроизведения
5	VDC	Режим воспроизведения постоянного напряжения с автоматическим выбором предела
6	VDC_R	Режим воспроизведения постоянного напряжения с ручным выбором предела
7	VAC	Режим воспроизведения переменного напряжения с автоматическим выбором предела
8	VAC_R	Режим воспроизведения переменного напряжения с ручным выбором предела
9	IDC	Режим воспроизведения постоянного тока с автоматическим выбором предела
10	IDC_ R	Режим воспроизведения постоянного тока с ручным выбором предела
11	IAC	Режим воспроизведения переменного тока тока с автоматическим выбором предела
12	IAC_R	Режим воспроизведения переменного тока тока с ручным выбором предела
13	R	Режим воспроизведения сопротивления
14	MODE	Выбор режима работы
15	TYPE, FUNK, FUNCTION	Выбор типа работы
16	RANGE	Режим предела

17	VALUE	Выбор значения
18	WIRE	Выбор двух/четырехпроводного режима в режиме измерения
19	KAL, KALC	Ввод калибровочной константы
20	KALCR, KALR	Ввод калибровочной константы сопротивления

DriverGet B1_28 – Захват данных.		
№	**Команда**	**Название**
1	VALUE, READ	Считывание показаний прибора

Подробное описание команд и примеры.

DriverSet B1_28 – отправка команд.			
Команда	**Описание**	**Аргумент**	**Пример**
STOP	Отключение выхода калибратора-вольтметра		DriverSet B1_28 STOP
GO	Подключение выхода калибратора-вольтметра		DriverSet B1_28 RESET
TIMEOUT	Максимальное время ожидания ответа от прибора	<численное значение, мс>	DriverSet B1_28 TIMEOUT=2500
CONN, ZCOMP	Выбор схемы подключения	2w	DriverSet B1_28 CONN=2W
		4w	
VDC	Режим воспроизведения постоянного напряжения с автоматическим выбором предела	<напряжение, В>	DriverSet B1_28 VDC 10
VDC_R	Режим воспроизведения постоянного напряжения с ручным выбором предела	<напряжение, В>, <предел, В>	DriverSet B1_28 VDC_R=5 10
VAC	Режим воспроизведения переменного напряжения с автоматическим выбором предела	<напряжение, В><частота, Гц>	DriverSet B1_28 VAC=10 400

VAC _ R	Режим воспроизведения переменного напряжения с ручным выбором предела	<напряжение, В><частота, Гц><предел, В><предел, Гц>	DriverSet B1_28 VAC_R=10 1000 10 1000
IDC	Режим воспроизведения постоянного тока с автоматическим выбором предела	<ток, А>	DriverSet B1_28 IAC=10m
IDC_ R	Режим воспроизведения постоянного тока с ручным выбором предела	<ток, А><предел, А>	DriverSet B1_28 IDC_R=10м 100м
IAC	Режим воспроизведения переменного тока с автоматическим выбором предела	<ток, А><частота, Гц>	DriverSet B1_28 IAC=0,01 100
IAC_R	Режим воспроизведения постоянного тока с ручным выбором предела	<ток, А><частота, Гц><предел, А><предел, Гц>	DriverSet B1_28 IAC_R=0,001 1000 0,01 100k
R	Режим воспроизведения сопротивления	<сопротивление, Ом>	DriverSet B1_28 R=100
MODE	Режим работы	GEN	DriverSet B1_28 MODE=GEN
		MEAS	
TYPE, FUNK, FUNCTION	Тип работы	VDC	DriverSet B1_28 FUNCTION=OHMS
		VAC	
		IDC	
		IAC	
		R, OHMS	
		KR, K_R, KILO_R	
RANGE	Выбор предела	AUTO, A	DriverSet B1_28 RANGE=A
		0,1	
		1	
		10,10k	

		100, 100k	
		1000, 1M	
		10M	
VALUE	Ввод значения	<значение>	DriverSet B1_28 VALUE=10,2314
WIRE	Выбор двух/четырехпроводного режима в режиме измерения	2	DriverSet B1_28 WIRE[i]=4
		4	
KAL, KALC	Ввод калибровочной константы	<значение>	DriverSet B1_28 KALC=10,2314
KALCR, KALR	Ввод калибровочной константы сопротивления	<значение>, Ом	DriverSet B1_28 KALCR=99,985

Приложение 12. Вольтметр В1-18

Перечень команд, доступных для вольтметра В1-18 с допустимыми параметрами.

DriverSet B1_18 – отправка команд		
№	**Команда**	**Описание**
1	STOP	Отключение выхода вольтметра
2	GO	Подключение выхода вольтметра
3	TIMEOUT	Выбор времени ожидания драйвера
5	MODE	Выбор режима работы
6	RANGE	Режим предела
7	VALUE	Выбор значения
8	FILETR_EN	Включение фильтра
9	SUBSET_EN	Режим вычитания
10	MULT_EN	Режим умножения
11	AVER_EN	Режим усреднения

DriverGet B1_18 – Захват данных.		
№	**Команда**	**Описание**
1	VALUE, READ	Считывание показаний прибора

Подробное описание команд и примеры.

DriverSet B1_18 – отправка команд.			
Команда	**Описание**	**Аргумент**	**Пример**
STOP	Отключение выхода калибратора-вольтметра		DriverSet B1_18 STOP
GO	Подключение выхода калибратора-вольтметра		DriverSet B1_18 RESET

TIMEOUT	Максимальное время ожидания ответа от прибора	<численное значение, мс>	DriverSet B1_18 TIMEOUT=2500
MODE	Режим воспроизведения постоянного напряжения	GEN, GENERATION	DriverSet B1_18 MODE=MEAS
	Режим измерения постоянного напряжения	MEASV, MEASUREMENTV, MEAS	
	Режим измерения постоянного напряжения относительно опорного	MEASDV	
RANGE	Выбор предела	10	DriverSet B1_18 RANGE=100
		100	
		1000	
VALUE	Ввод значения	<значение>	DriverSet B1_18 VALUE=10,2314
FILETR_EN	Включение фильтра	ON	DriverSet B1_18 FILTER_EN=ON
		OFF	
MULT_EN	Включение операции умножения	ON <значение>	DriverSet B1_18 MULT_EN =ON 1,2 DriverSet B1_18 MULT_EN =OFF
		OFF	
AVER_EN	Включение операции усреднения	ON <значение>	DriverSet B1_18 AVER_EN =ON 7 DriverSet B1_18 AVER_EN =OFF
		OFF	
SUBSET_EN	Включение операции вычитания	ON <значение>	DriverSet B1_18 SUBSET_EN =ON 1,3 DriverSet B1_18 SUBSET _EN =OFF
		OFF	

Приложение 13. Вольтметр Fluke 8508A

Перечень команд, доступных для вольтметра Fluke 8508A с допустимыми параметрами.

DriverSet Fluke8508A – отправка команд		
№	**Команда**	**Описание**
1	RESET	Перезагрузка и сброс всех параметров
2	GUARD	Выбор защиты для всех функций
3	TIMEOUT	Выбор времени ожидания драйвера
4	FUNCTION, FUNC	Конфигурирование мультиметра для входного параметра
5	RANGE	Выбор предела
6	SETTINGS, SET	Установка параметров мультиметра
7	INPUT	Конфигурации входа и расчета отношения сигналов

DriverGet Fluke8508A – Захват данных.		
№	**Команда**	**Описание**
1	READ	Считывание последнего измеренного значения
2	FETCH	Мгновенное считывание значения отображаемого индикатором

Подробное описание команд и примеры.

DriverSet Fluke8508A – отправка команд				
Команда	**Описание**	**Аргумент**	**Описание аргумента**	**Пример**
RESET	Перезагрузка и сброс всех параметров			DriverSet Fluke8508A RESET
GUARD	Выбор защиты для всех функций	INT	Выбор варианта подключения защиты мультиметра: внутренняя или внешняя.	DriverSet Fluke8508A GUARD=INT
		EXT		

TIMEOUT	Максимальное время ожидания ответа от прибора	<численное значение, мс>		DriverSet Fluke8508A TIMEOUT=2500
FUNCTION, FUNC	Конфигурирование мультиметра для входного параметра	VDC	Режим постоянного напряжения	DriverSet Fluke8508A FUNC=IAC
		VAC	Режим переменного напряжения	
		IDC	Режим постоянного тока	
		IAC	Режим переменного тока	
		OHMS, R	Режим измерения нормального значения сопротивления	
		HIV_OHMS, HIV_R	Режим измерения сопротивления на высоком напряжении	
		TRUE_OHMS, TRUE_R	Режим измерения истинной величины сопротивления	
RANGE	Выбор предела	<значение>		DriverSet Fluke8508A RANGE=20
SETTINGS, SET	Установка параметров мультиметра	RANGE_AUTO, AUTO	Автоматический выбор предела измерения	DriverSet Fluke8508A SET = RANGE_AUTO
		FILTER_ON, FILT_ON	Включение/ выключение аналогового фильтра на пути сигнала	
		FILTER_OFF, FILT_OFF		
		RESL_5, RESOLUTION_5, RESL5	Установка разрешающей способности измерений соответственно 5, 6, 7, 8 цифр	
		RESL_6, RESOLUTION_6, RESL6		
		RESL_7, RESOLUTION_7, RESL7		

		RESL_8, RESOLUTION_8, RESL8		
		FAST_ON	Включение/ выключение режима ускоренных измерений	
		FAST_OFF		
		TWO_WR, 2W	Выбор схемы измерения	
		FOUR_WR, 4W		
		FILT100HZ, FILTER_100HZ	Выбор фильтра для среднеквадратичного преобразования, что позволяет производить измерения на частотах до выбранной	
		FILT40HZ, FILTER_40HZ		
		FILT10HZ, FILTER_10HZ		
		FILT1HZ, FILTER_1HZ		
		TFER_ON	Ускорение считывания данных за счет точности результатов	
		TFER_OFF		
		DC, DCCP	Выбор измерений со связью по постоянному току	
		AC, ACCP	Выбор измерений со связью по переменному току	
		SPOT_ON, FREQ_CORR_ON	Включение/ выключение поправки по частоте	
		SPOT_OFF, FREQ_CORR_OFF		
		LOW_CURR_ON, LOI_ON	Включение/ выключение режима измерения на малом токе	
		LOW_CURR_OFF, LOI_OFF		
INPUT	Конфигурации входа и расчета	FRONT	Выбор разъемов передней панели	DriverSet Fluke8508A

	отношения сигналов	REAR	Выбор разъемов задней панели	INPUT=FRONT
		DIV_REAR	Выбор измерений с передних разъемов с последующими измерениями с задних разъемов и расчетом отношения сигналов Front/Rear	
		SUB_REAR	Выбор измерений с передних разъемов с последующими измерениями с задних разъемов и расчетом разницы между сигналами Front-Rear	
		DEVTN, DEVIATION	Выбор измерений с передних разъемов с последующими измерениями с задних разъемов и расчетом отношения сигналов Front-(Front/Rear)	
		OFF	Отключение всех разъемов и отмена измерений со сканированием	

Приложение 14. Калибратор Fluke 55xx и 57xx

Перечень команд, доступных для калибратора Fluke серий 55xx и 57xx с допустимыми параметрами.

DriverSet 5XXX – отправка команд		
№	**Команда**	**Описание**
1	STOP	Отключение выхода калибратора
2	RESET	Перезагрузка и сброс всех параметров
3	CONN	Выбор схемы подключения
4	TIMEOUT	Таймаут
5	VDC	Режим постоянного напряжения
6	VAC	Режим переменного напряжения
7	IDC	Режим постоянного тока
8	IAC	Режим переменного тока
9	R	Режим сопротивления
10	C	Режим емкости
11	F	Режим частоты
12	TC	Режим термопары

Подробное описание команд и примеры.

DriverSet 5XXX – отправка команд.			
Команда	**Описание**	**Аргумент**	**Пример**
STOP	Отключение выхода калибратора		DriverSet 5XXX STOP
RESET	Перезагрузка и сброс всех параметров		DriverSet 5XXX RESET
CONN (ZCOMP)	Выбор схемы подключения	2w	DriverSet 5XXX CONN=2w
		4w	

		NONE (NO)	
TIMEOUT	Максимальное время ожидания ответа от прибора	<численное значение, мс>	DriverSet 5XXX TIMEOUT=2500
VDC	Режим постоянного напряжения	<напряжение, В>	DriverSet 5XXX VDC=5
VAC	Режим переменного (синусоидального) напряжения	<напряжение, В><частота, Гц>	DriverSet 5XXX VAC=10 400
IDC	Режим постоянного тока	<ток, А><тип выхода> Типы выхода: A20, AUX	DriverSet 5XXX IDC=500m aux
IAC	Режим переменного тока	<ток, А><частота, Гц><тип выхода> Типы выхода: A20, AUX	DriverSet 5XXX IAC=100m 50 aux
R	Режим сопротивления	<сопротивление, Ом>	DriverSet 5XXX R=100k
C	Режим емкости	<емкость, Ф>	DriverSet 5XXX C=5u
F	Режим частоты	<частота, Гц><размах напряжения, В>	DriverSet 5XXX F=10k 5
TC	Режим термопары	<температура, °><тип термопары><тип шкалы> Тип термопары: Тип шкалы: TS90, TS68	DriverSet 5XXX TC=30 J TS90