ООО «Техноград плюс»

ТЕХНОГРАД. СИСТЕМА СБОРА ДАННЫХ И УПРАВЛЕНИЯ (ТЕХНОГРАД ССДУ)

Подсистема сценариев модуля ШПД

РУКОВОДСТВО АДМИНИСТРАТОРА

Новосибирск

2023

Оглавление

Термины, их сокращения и определение

Административный интерфейс – раздел системы, содержащий интерфейс пользователя для управления модулями системы, структурой, содержанием, пользователями и другими составляющими модуля.

БД – база данных

Время ожидания – время, которое команда находилась в очереди на выполнение

Время выполнения – время выполнения команды

ТЕХНОГРАД ССДУ – ПО «ТЕХНОГРАД. СИСТЕМА СБОРА ДАННЫХ И УПРАВЛЕНИЯ (ТЕХНОГРАД ССДУ))» ООО «Техноград плюс» (г. Новосибирск)

Драйвер – компонент ТЕХНОГРАД ССДУ, обеспечивающий взаимодействие с сетевым элементом

Команда – задача, выполняемая ТЕХНОГРАД ССДУ по отношению к удалённому оборудованию

Конфигурация – набор параметров, включающих в себя: драйвер, авторизационные данные, профили подсистемы сбора данных и планировщика, дополнительные (задаются администратором) и специальные параметры (принудительная маршрутизация

МУИК – Модуль «Управляющий Измерительный Комплекс» «Графической информационной системы СВЯЗЬ (ГИС СВЯЗЬ)» – предыдущее наименование ТЕХНОГРАД ССДУ.

Оператор ТП – оператор технической поддержки

Подсистема сбора данных (Мониторинг) – подсистема ТЕХНОГРАД ССДУ для автоматического выполнения команд с заданной периодичностью и накоплением результатов их выполнения.

ПП – программный продукт

СЭ (*Сетевой элемент*) – разнородное оборудование ШПД и сервисные платформы, предоставляющие услуги абонентам.

Сценарии – последовательность MML-инструкций, предназначенных для выполнения на сетевом элементе, а также специальных директив для удобства создания MML-инструкций и управления их выполнением.

Тестирование команды – отправка команды от имени администратора для проверки процесса выполнения, времени ожидания, времени выполнения команды и так далее.

1 Введение

Полное наименование программного продукта (ПП): ТЕХНОГРАД. СИСТЕМА СБОРА ДАННЫХ И УПРАВЛЕНИЯ (ТЕХНОГРАД ССДУ). В целях дальнейшей идентификации программного продукта в настоящем документе используются его краткие наименования: ТЕХНОГРАД ССДУ, Система.

ТЕХНОГРАД. СИСТЕМА СБОРА ДАННЫХ И УПРАВЛЕНИЯ (ТЕХНОГРАД ССДУ) представляет собой систему сбора и обработки данных, позволяющую автоматизировать процесс взаимодействия с разнородным оборудованием. Система предназначена для централизации и унификации работы операторов службы технической поддержки, связанной с диагностикой неисправностей абонентских линий, управлением услугами абонентов и прочими задачами технической эксплуатации.

Руководство администратора Подсистемы сценариев модуля ШПД ТЕХНОГРАД ССДУ (далее – «Руководство») предназначено для специалистов (далее – «Администратор ШПД»), производящих разработку и выполнение сценариев взаимодействия Системы с оборудованием с использованием web-интерфейса.

В целях дальнейшей идентификации Подсистемы сценариев модуля ШПД ТЕХНОГРАД ССДУ в настоящем документе используются её краткое наименование Подсистема.

2 Принципы устройства и функционирования, основные возможности

Подсистема позволяет создавать, хранить и редактировать сценарии – последовательность MML-инструкций, предназначенных для выполнения на сетевом элементе, а также специальных директив для удобства создания MML-инструкций и управления их выполнением.

Сценарий может соответствовать определенной конфигурации модуля ШПД для возможности его выполнения на сетевом элементе с помощью модуля ШПД (см. Руководство Администратора ШПД).

Подсистема состоит из набора элементов, все из которых (кроме Сценария и Переменной) имеют свойства каталогов, т.е. могут содержать в себе любые другие элементы. Также Подсистема имеет специальные типы элементов для решения задач активации и проверки технических и абонентских сервисов.

Подсистема включает в себя:

- Репозиторий сценариев – базу данных шаблонов скриптов. Сценарии хранятся в Подсистеме в виде древовидной структуры, что позволяет их структурировать, а также создавать общие сценарии для нескольких конфигураций модуля ШПД. Подсистема изначально имеет предопределенную структуру в виде разделения сценариев по типам и производителям сетевых элементов, но её допускается расширять и изменять.
- Подсистему администрирования шаблонов скриптов – пользовательский web-интерфейс для доступа к БД шаблонов скриптов, создания и модификации шаблонов скриптов.
- Командный язык для конфигурационных скриптов активного сетевого оборудования мультисервисных сетей.
- Препроцессор обработки инструкций командного языка конфигурационных скриптов на конечном оборудовании и пошаговый интерпретатор.

3 Начало и завершение работы с Подсистемой

Работа Администратора ШПД с Подсистемой осуществляется в пользовательском web-интерфейсе.

- Начало работы – переход к интерфейсу Подсистемы – производится из административного интерфейса модуля ШПД по ссылке «Сценарии» (см. рис. 1). Об административном интерфейсе модуля ШПД подробнее см. Руководство Администратора ШПД.

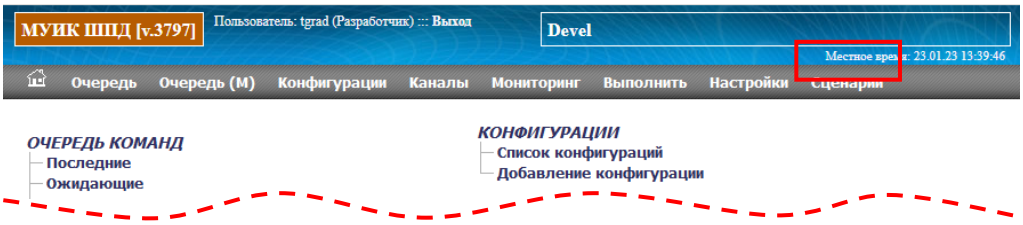

Рисунок 1 – Размещение ссылки перехода в Подсистему из административного интерфейса модуля ШПД

- При переходе по ссылке открывается окно пользовательского интерфейса Подсистемы (см. п. 4), и оно становится активным. Исходное окно административного интерфейса модуля ШПД при таком переходе не закрывается.
- Завершение работы производится закрытием соответствующего окна (вкладки) браузера, или переходом по ссылке для выхода из Подсистемы (см. рис. 2).

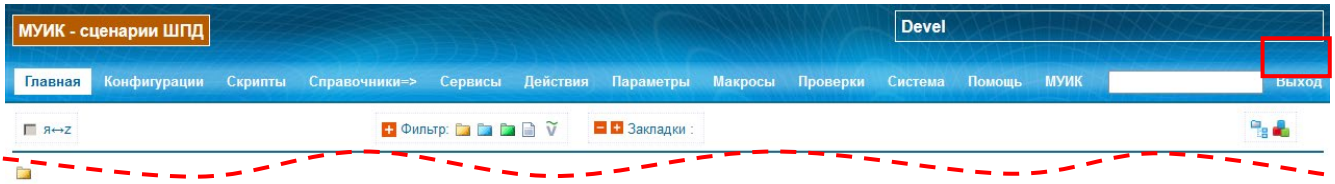

Рисунок 2 – Размещение ссылки для выхода из Подсистемы в интерфейсе

4 Пользовательский интерфейс, элементы Подсистемы

4.1 Структура экранной формы интерфейса пользователя

Рисунок 3 содержит пример экранной формы интерфейса пользователя.

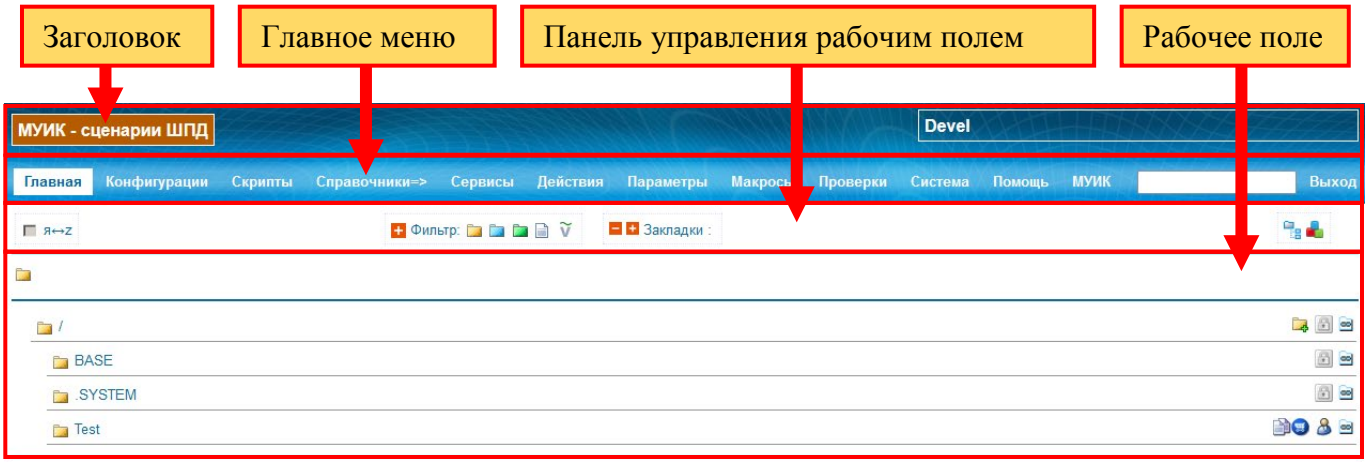

Рисунок 3 – Пример экранной формы Подсистемы

Рисунок 4 содержит описание структуры заголовка.

Рисунок 4 – Структура заголовка

Настраиваемое описание сервера – аналогично тому, которое выводится в административном интерфейсе модуля ШПД (см. Руководство Администратора ШПД).

Разделы главного меню и их назначение:

- Разделы, в рабочее поле которых выводятся элементы репозитория сценариев в древовидной структуре:
  - **Главная** – представление всей структуры репозитория сценариев.
  - **Конфигурации** – представление структуры репозитория с отображением каталогов и конфигураций по пути /BASE.
  - **Скрипты** – переход к структуре каталога /.SYSTEM/BASE_COMMANDS, предназначен для хранения набора базовых скриптов.
  - **Справочники** – переход к структуре каталога /.SYSTEM, предназначен для хранения системных и пользовательских справочников.

- **Сервисы** – переход к структуре каталога /.SYSTEM/.COMMANDS, предназначен для хранения справочника имен сервисов.
- **Действия** – переход к структуре каталога /.SYSTEM/.ACTIONS, предназначен для хранения справочника имен сценариев.
- **Параметры** – переход к структуре каталога /.SYSTEM/.PARAMS, предназначен для хранения справочника имен переменных для сценариев.
- **Макросы** – переход к структуре каталога /.SYSTEM/.EDIT/.MACROS, предназначен для хранения справочника сценариев с предопределенными функциями для использования в сценариях.

- **Проверки** – открытие формы «Мастер создания сценариев» (см. п. 6.3). Описание работы со списками проверок сервисов см. в п. 6.
- **Система** – переход к редактированию системного шаблона 'root' (см. п. 4.5.10).
- **Помощь** – ссылка для открытия окна с краткой информацией по описанию языка сценариев (см. рис. 5).

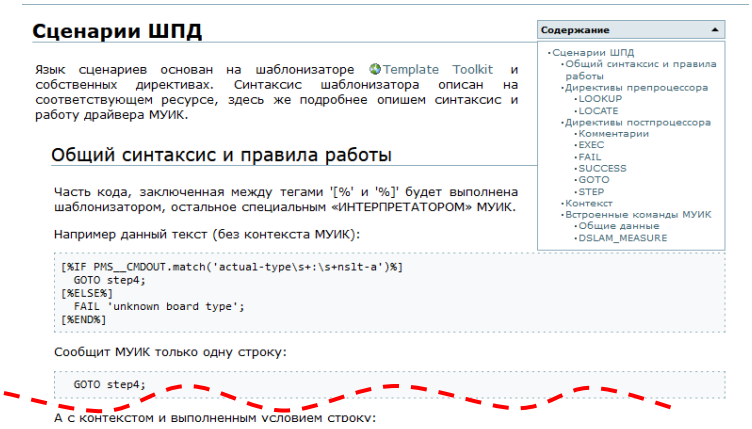

Рисунок 5 – Образец содержимого окна помощи по языку сценариев

- **МУИК** – ссылка для открытия административного интерфейса модуля ШПД. При переходе по ссылке открывается окно интерфейса модуля ШПД и оно становится активным (фокус переходит к нему). Исходное окно административного интерфейса Подсистемы при таком переходе не закрывается.
- **Поле ввода** – поле ввода текстового параметра для фильтра. После ввода искомого выражения по клавише Enter будет осуществлен поиск вхождения введенной последовательности символов в названиях и описаниях элементов дерева и вывод найденного на экран.

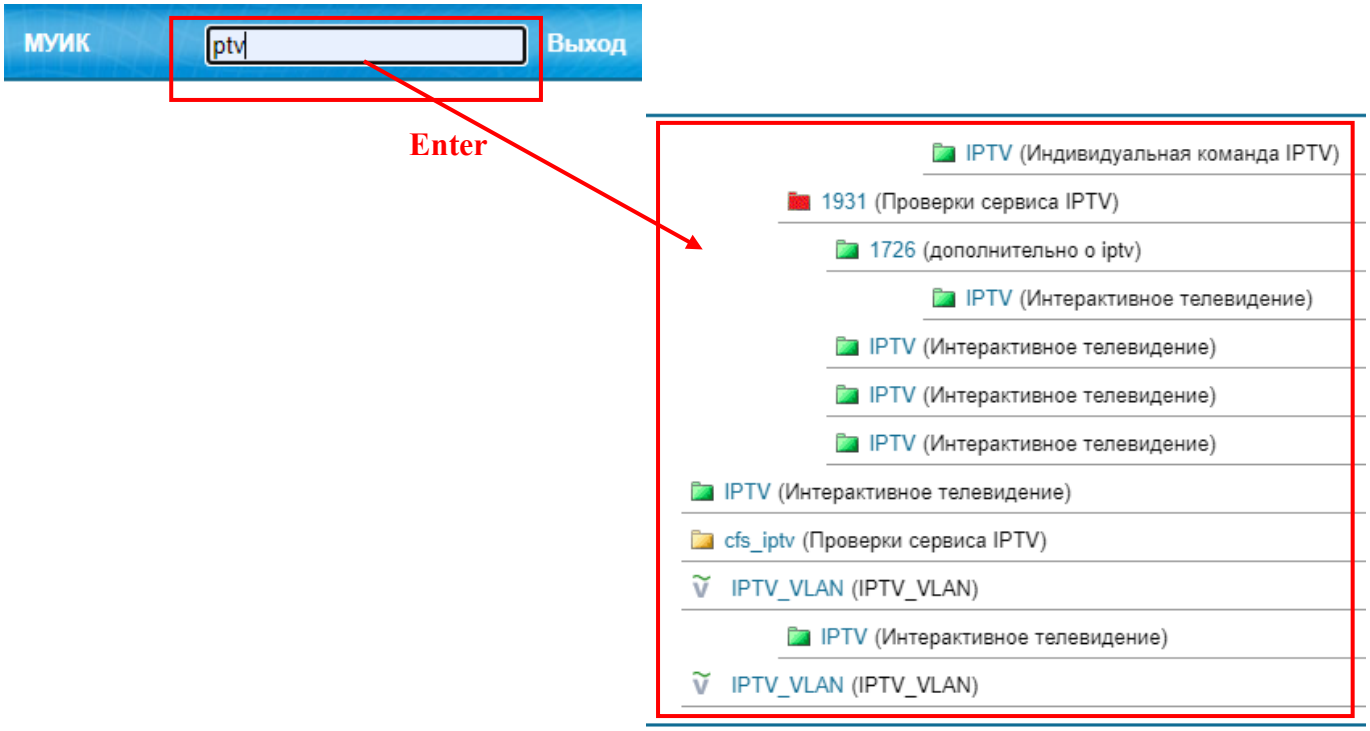

Рисунок 6 – Пример работы фильтра по критерию поиска «ptv»

- **Выход** – выход из административного web-интерфейса и закрытие окна.

4.2 Типы элементов Подсистемы

Типы элементов Подсистемы и их обозначения (пиктограммы):

– каталог. Элемент может содержать любые другие элементы. Используется только для группировки прочих элементов.

(каталог, помеченный сиволом ★) – конфигурация (модели). Используется при поиске сценария в Подсистеме.

– базовая конфигурация. Специальный элемент, имя которого должно соответствовать одной из конфигураций модуля ШПД. Используется при поиске сценария в Подсистеме. Для правильной работы системы базовая конфигурация должна находится в корне конфигурации модели. Базовых конфигураций в конфигурации модели может быть несколько.

– сервис. Специальный элемент, используется при поиске сценария в Подсистеме.

– список проверок (кратко - Проверки). Специальный элемент, используется при поиске списка сценариев проверок в Подсистеме.

Примечание. *Общее свойство элементов типов «сервис» и «список проверок» заключается в том, что они содержат сценарии. Принципиальное их различие*

состоит в том, что по алгоритму поиска и подготовки сценария к выполнению при поиске сценария сервиса возвращается один сценарий, а при поиске списка проверок возвращается список сценариев. Подробнее см. п. 5.

– сценарий. Элемент, который содержит код сценария.

– переменная. Элемент, который содержит описание и значение по умолчанию для любой произвольной переменной, используемой в сценариях.

Формальных правил иерархической подчинённости элементов друг другу нет, но при размещении элементов обязательно следует учитывать алгоритм поиска и подготовки сценария к выполнению (см. п. 5).

4.3 Панель управления рабочим полем

4.3.1 Структура панели

Структура панели управления рабочим полем приведена на следующей иллюстрации.

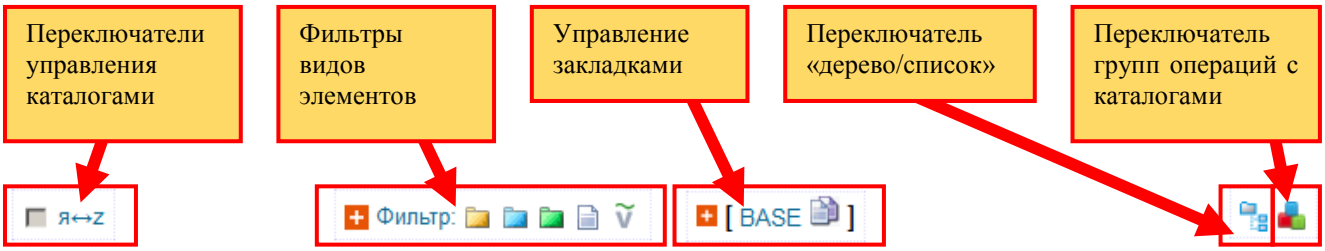

Рисунок 7 – Структура панели

Если рабочее поле находится в режиме «Проверки», или «Система» (см. главное меню), или редактирования шаблонов, то активация таких элементов, как:

- элементов панели «Переключатели управления каталогами»,
- переключатель «дерево/список»,
- переключатель групп операций с каталогами

приводит к переводу отображения информации в режим «Главная».

4.3.2 Переключатели управления каталогами

4.3.2.1 Показать/спрятать управление

Выбор «показать/спрятать управление» происходит по активации переключателя «» (см. рис. 8).

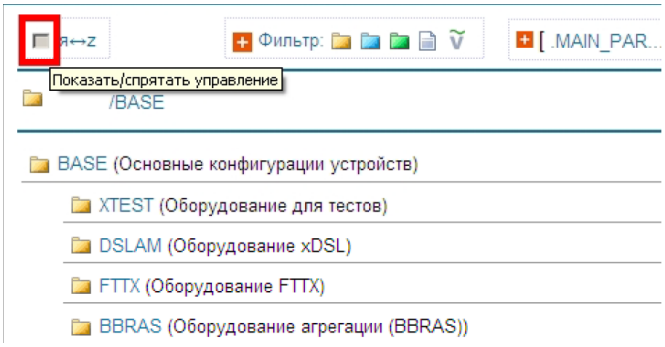

Рисунок 8 – Переключатель «показать/спрятать управление»

Результат активации элемента «показать/спрятать управление» показан на рисунке 9: переключается между режимами без элементов управления каталогами и с ними.

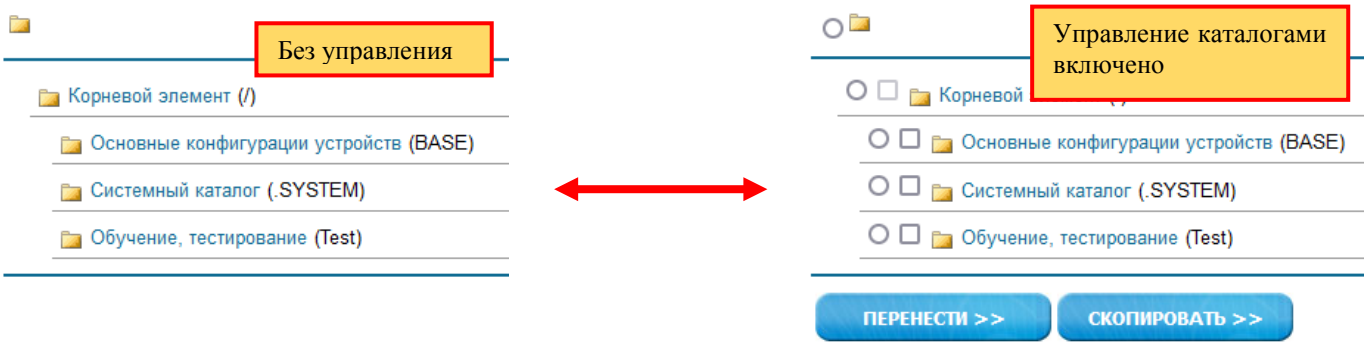

Рисунок 9 – Переключение отображения «показать/спрятать управление»

В режиме управления каталогами:

- Флагами выбора ☑ отмечаются элементы дерева репозитория (один или несколько), которые необходимо перенести или копировать.
- Переключателем ◉ отмечается каталог (один), куда необходимо копировать или перенести отмеченные флагом выбора.
- Кнопками ПЕРЕНЕСТИ >> СКОПИРОВАТЬ >> выполняются операции переноса или копирования.

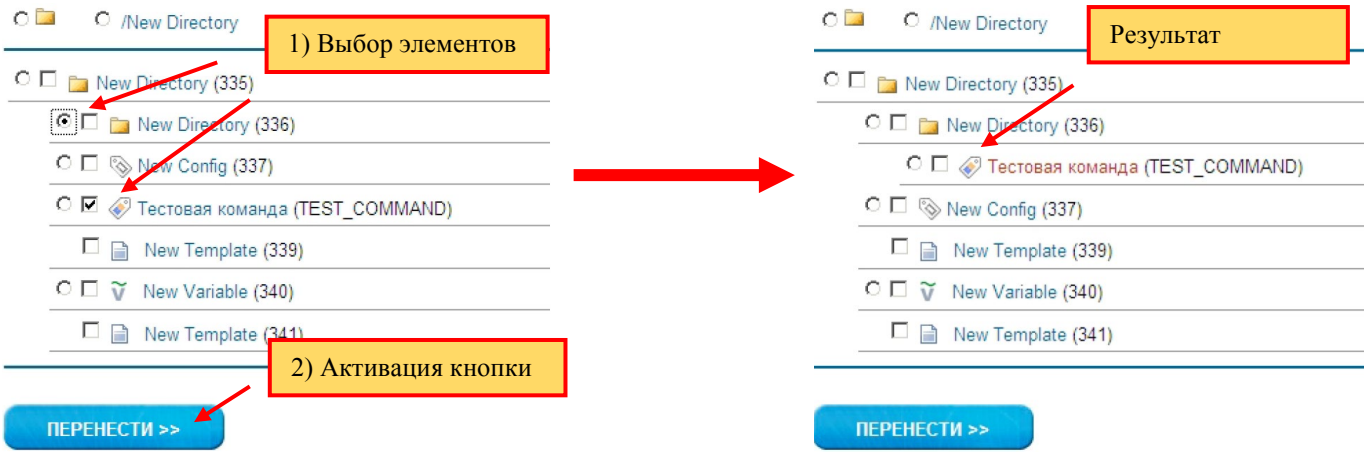

Рисунок 10 – Пример переноса

В отличие от копирования в буфер (элемент управления -) производится копирование выделенных каталогов вместе с их вложением. При этом целевой каталог не может быть вложен в копируемый.

4.3.2.2 Выбор режима отображения

Отображение дерева репозитория имеет несколько режимов, отличающихся комбинациями сочетания заголовков элементов и их описания:

- отображать только наименование,
- отображать только описание,
- отображать наименование, а затем описание,
- отображать описание, а затем наименование.

Выбор происходит циклически по активации переключателя «Я↔Z» (см. рис. 11).

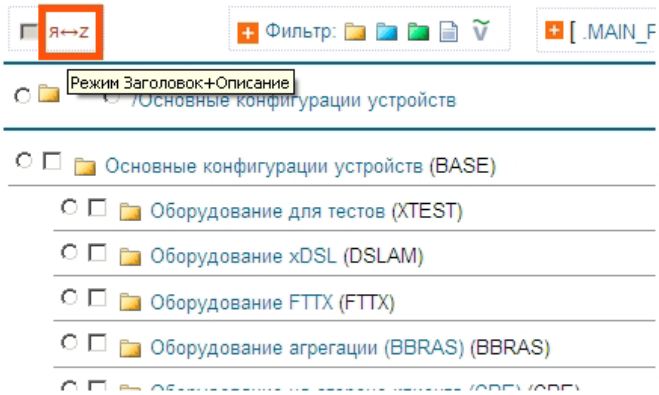

Рисунок 11 – Размещение переключателя выбора режима отображения и пример отображения в режиме «описание + наименование»

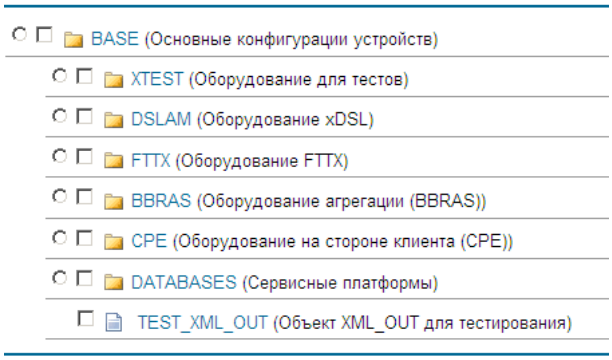

Рисунок 12 – Пример отображения в режиме «наименование + описание»

Рисунок 13 – Примеры отображения «только название» и «только описание»

4.3.3 Фильтры типов элементов

Набор фильтров типов элементов репозитория приведен на рисунке 14.

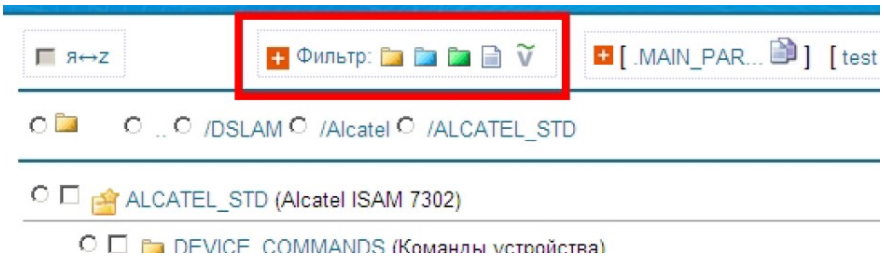

Рисунок 14 – Фильтры видов элементов репозитория

Элементы фильтра:

Фильтр: – отображать всё,

– отображать только разделы,

– отображать только конфигурации,

– отображать только сервисы,

– отображать только сценарии,

– отображать только параметры.

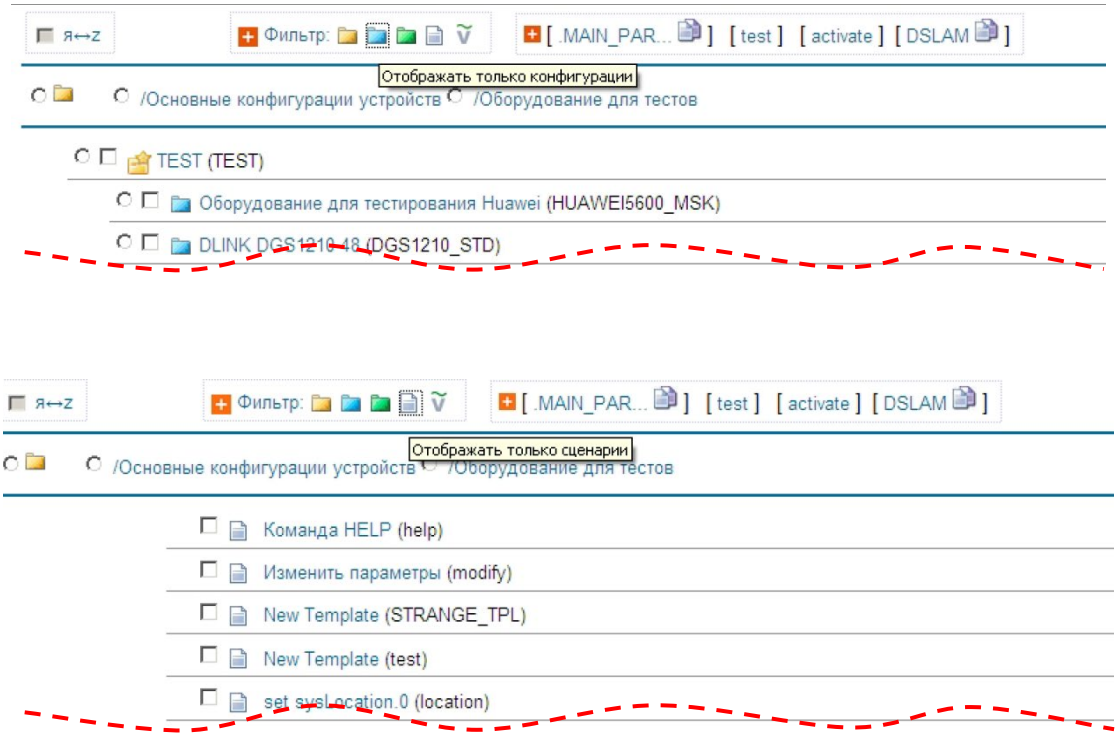

Рисунок 15 – Примеры применения фильтров

4.3.4 Управление закладками

Инструмент «Закладки» служит для быстрого доступа к элементу дерева, который был «запомнен» в закладке.

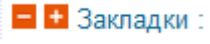

Рисунок 16 – Элементы управления закладками

Для создания закладки необходимо активировать пиктограмму . В списке закладок будет сохранена ссылка на текущий элемент репозитория (см. рис. 17).

Последняя добавленная закладка отображается левее других. Количество закладок ограничено четырьмя. Если количество закладок 4, то при добавлении новой закладки самая правая (добавленная ранее всех) исчезает.

В закладках рядом с наименованием присутствует управляющий элемент «вставить из буфера», что позволяет использовать предварительно скопированные в буфер элементы репозитория, не переходя в саму закладку. Если тип элемента репозитория в закладке не предполагает операции копирование в него, значок «вставить из буфера» отсутствует (см. рис. 17).

Рисунок 17 – Пример наполненной панели закладок и размещения элемента «копировать из буфера»

Для быстрого перехода к нужному элементу необходимо активировать соответствующею закладку (см. рис. 18). Активирование элемента приводит к удалению всего списка закладок.

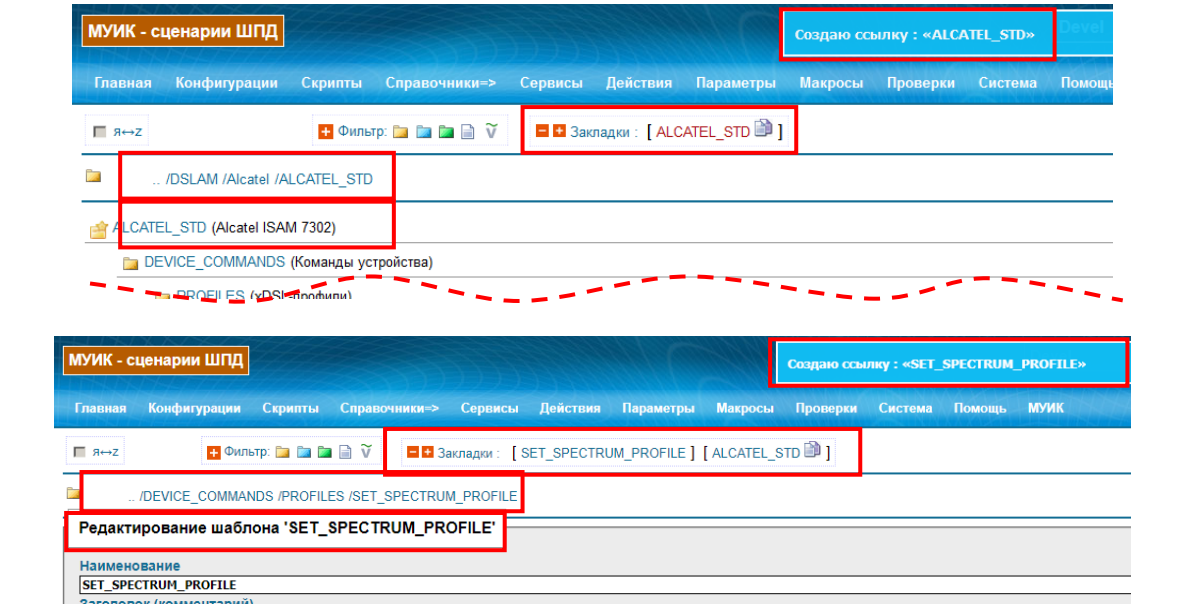

Рисунок 18 – Примеры создания закладки (вид сразу после активизации элемента создания закладки)

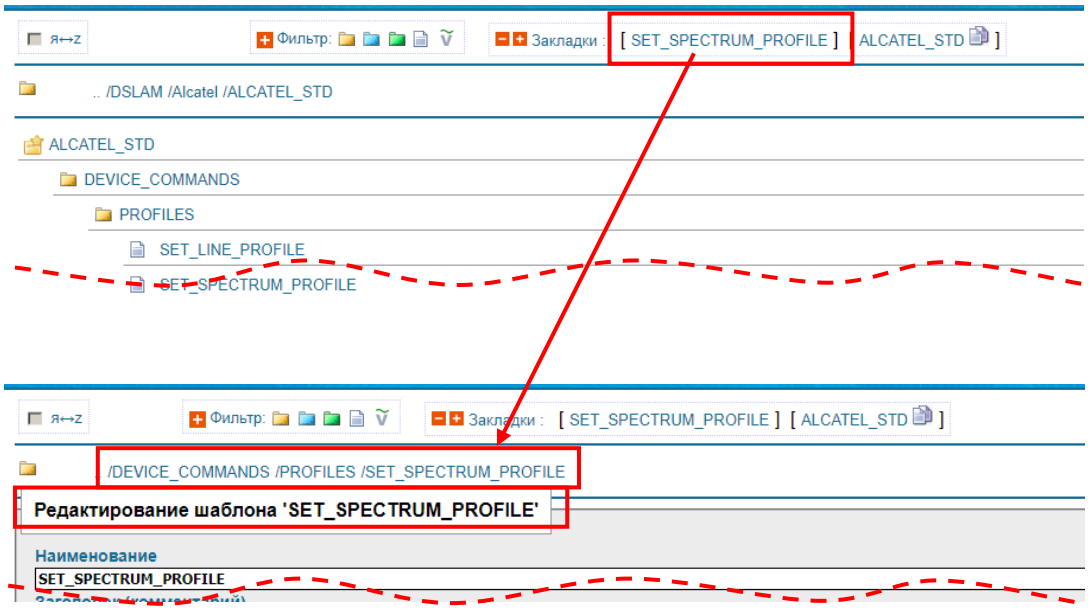

Рисунок 19 – Пример перехода по ссылке в закладке

4.3.5 Переключатель «дерево/список»

Переключатель «дерево/список» приведен на рисунке 20.

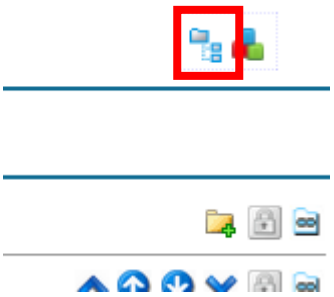

Рисунок 20 – Переключатель «дерево/список»

Активация переключателя переводит режим отображения структуры репозитория из вида дерева каталогов в просмотр дерева со списками вложенных элементов и обратно.

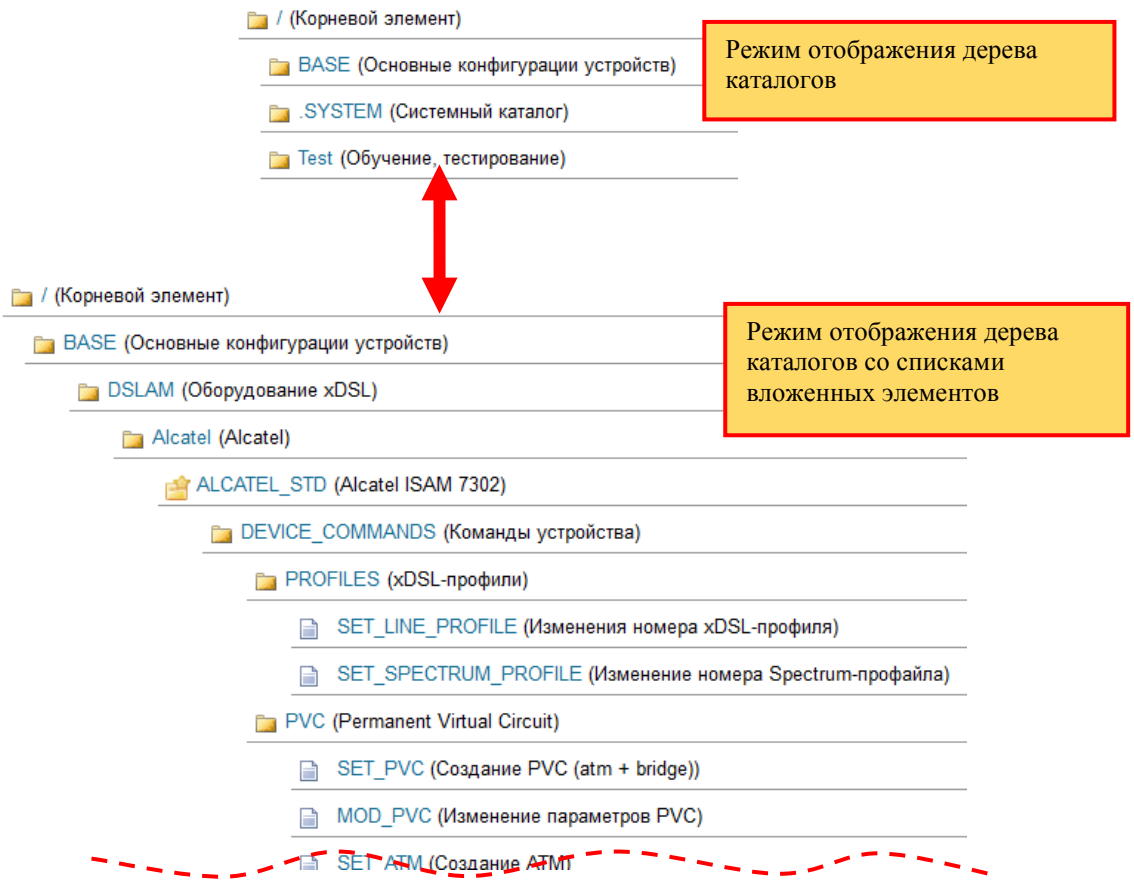

Рисунок 21 – Переключение режимов отображения репозитория

4.3.6 Переключатель групп операций с элементами репозитория

Переключатель групп операций с элементами репозитория приведен на рисунке 22.

Рисунок 22 – Переключатель групп операций

По активации переключателя отображается от одной до трех групп операций с элементами репозитория в различной комбинации по циклу, как показано на рисунке 22.

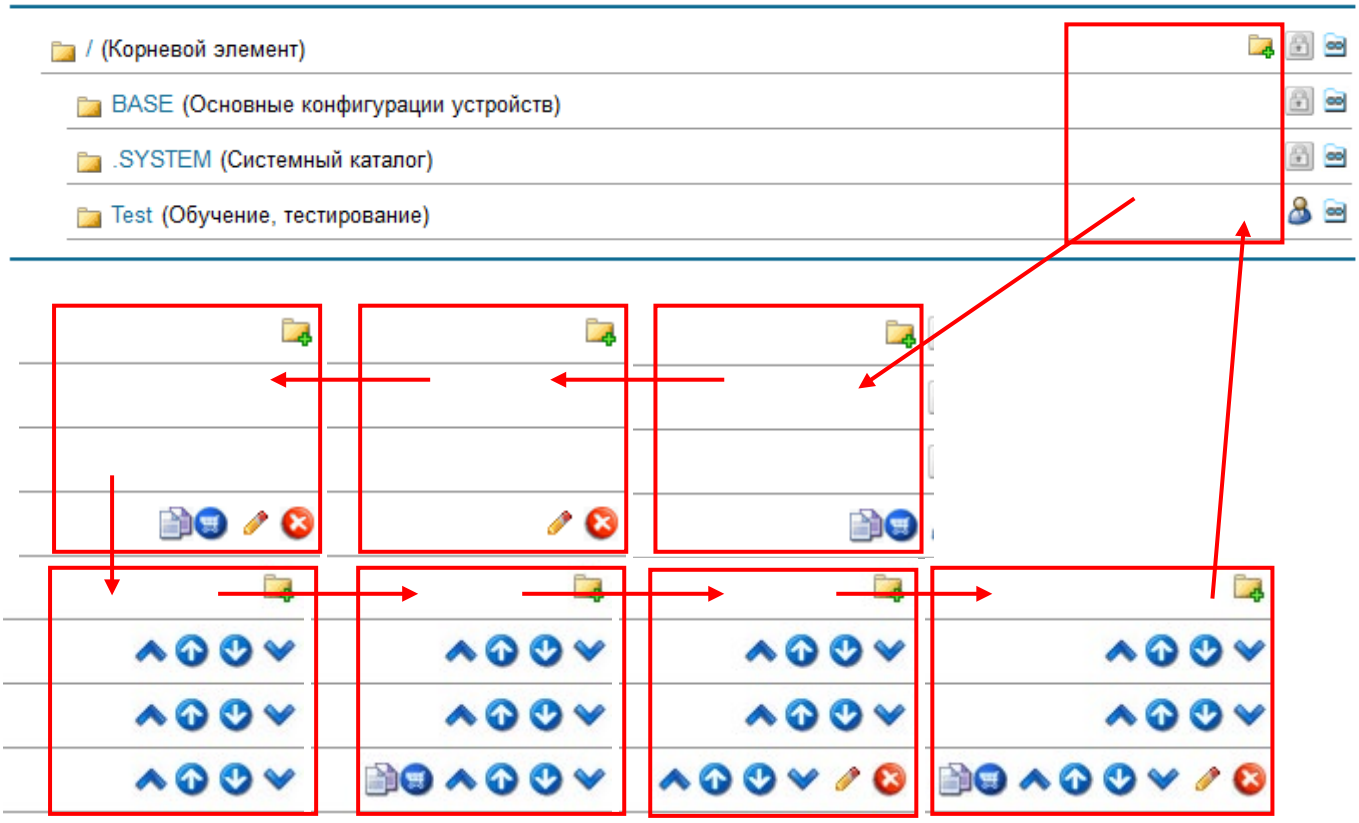

Рисунок 23 – Цикличность переключения групп операций

Операция «создать подраздел» в данном примере представляет группу операций добавления элементов репозитория, см. 4.5.2.

Группы операций над элементами:

- Группа «Copy/Paste»:
  - запомнить в буфере (каталоги запоминаются без вложений);
  - вставить из буфера.
- Подгруппа «Редактирования»:
  - редактировать;
  - удалить.
- Подгруппа «Перемещение»:
  - переместить в самый верх списка;
  - переместить на строку выше;
  - переместить на строку ниже;
  - переместить в самый низ списка.

Не все группы операций доступны для всех элементов репозитория. Отображение операций для каждого элемента производится в соответствии с его назначением и функциями.

4.4 Атрибуты элементов репозитория – «принадлежность» и «видимость»

Атрибуты элементов репозитория (см. рис. 23), выводимые в рабочее поле Подсистемы:

- принадлежность (кто создатель элемента):
  - – системный (элемент был предустановлен в репозитории);
  - – пользовательский (элемент был создан пользователем Подсистемы).

  Атрибут принадлежности недоступен для изменения.

- видимость (доступность) элемента во внешних системах (внешних по отношении к Подсистеме):
  - – элемент доступен только в Подсистеме, для других систем скрыт;
  - – доступен для выбора в меню экранных форм интерфейса оператора технической поддержки (см. Руководство оператора технической поддержки);
  - – видим для всех.

  Администратор ШПД в экранной форме Подсистемы может изменять атрибут видимости его активацией.

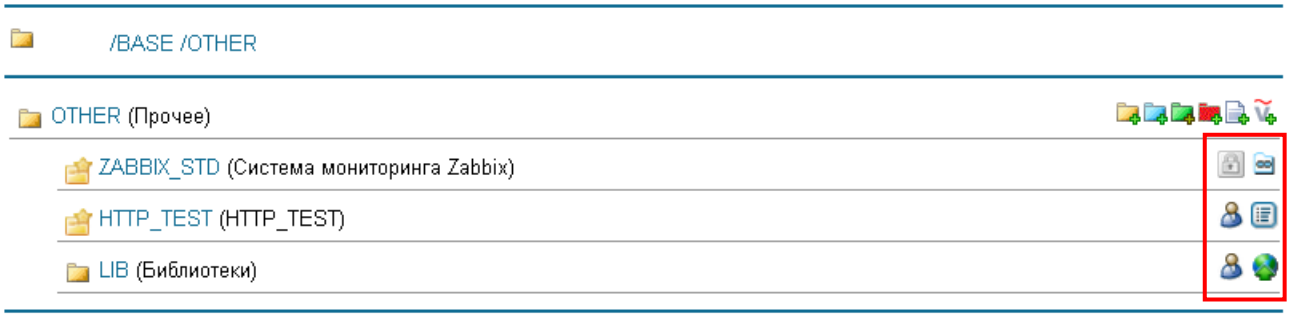

Рисунок 24 – Расположение в рабочем поле атрибутов принадлежности и видимости

4.5 Операции с элементами репозитория сценариев при отображении древовидной структуры

4.5.1 Навигация по дереву

Варианты перехода по дереву:

- Раскрыть дерево переключателем «дерево/список» (см. п. 4.3.5). В этом случае, поиск производится прокруткой дерева в экранной форме.
- Раскрыть ветку, активируя надпись папки, отображаемую ссылкой, содержащую нужную ветку (см. п. 4.3.2.2).
- Активировать необходимый элемент пути, выводимый первой строкой в рабочем поле (см. рис. 23).

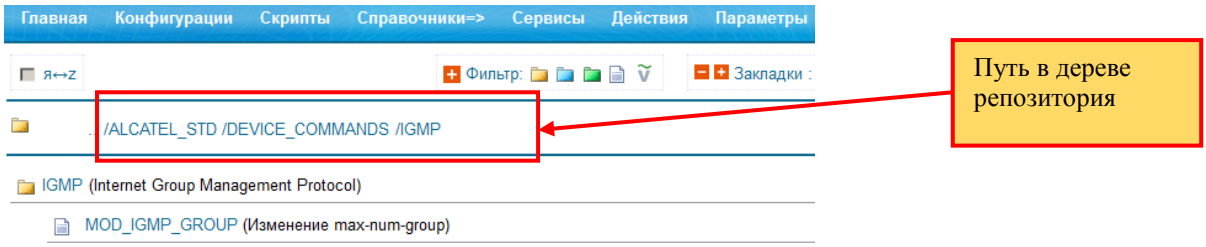

Рисунок 25 – Размещение в экранной форме пути по дереву репозитория

4.5.2 Добавление элемента репозитория

Группа элементов управления «Создать» содержит 5 элементов:

- создать подраздел,
- создать конфигурацию,
- создать сервис – сервис для конфигурации,
- создать сценарий – сценарий, действие над сервисом,
- создать переменную – параметр.

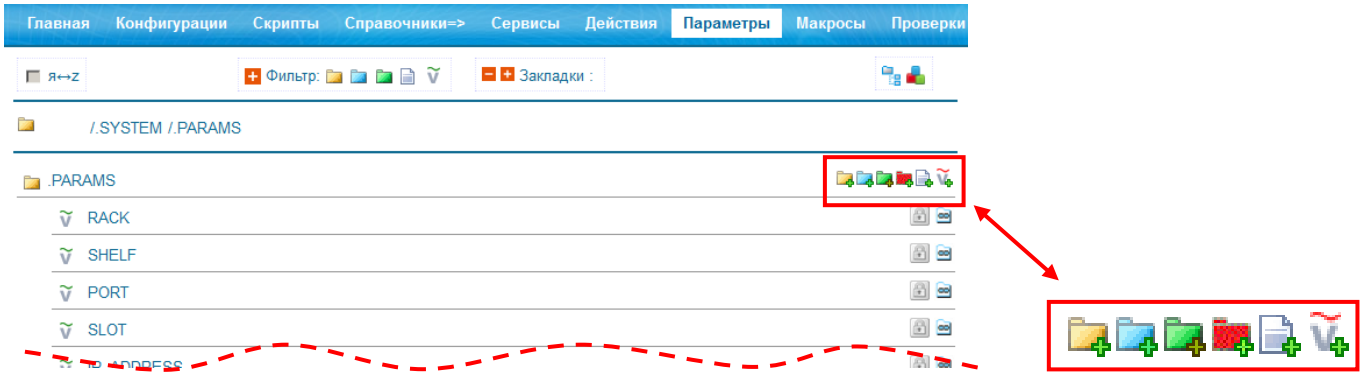

Рисунок 26 – Группа элементов управления «Создать»

Отображение элементов группы «Создать» производится в зависимости от типа текущего элемента.

***Пример**. Если текущий элемент является сервисом, будет отображено всего два элемента вида «Создать»: создать сценарий, создать переменную.*

Все элементы создаются с заголовком вида «New….» и наименованием, содержащим случайное число. Наименование и заголовок можно отредактировать. Для редактирования элементов дерева необходимо активировать пиктограмму «Редактировать», находящуюся напротив каждого элемента иерархии, либо активировать пиктограмму текущего элемента. Пример формы редактирования одного из элементов приведен на рис. 27.

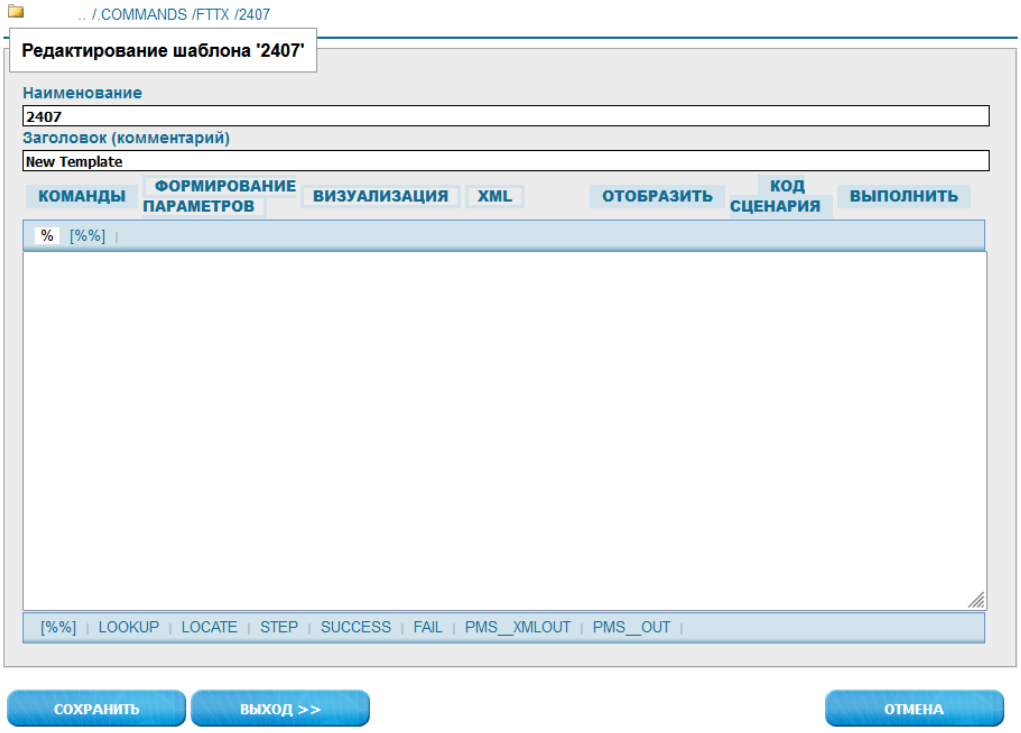

Рисунок 27 – Пример экранной формы редактирования нового элемента с наименованием «2407»

4.5.3 Удаление элемента

Для удаления элемента из иерархии необходимо активировать пиктограмму в строке выбранного элемента, затем выполнить подтверждение.

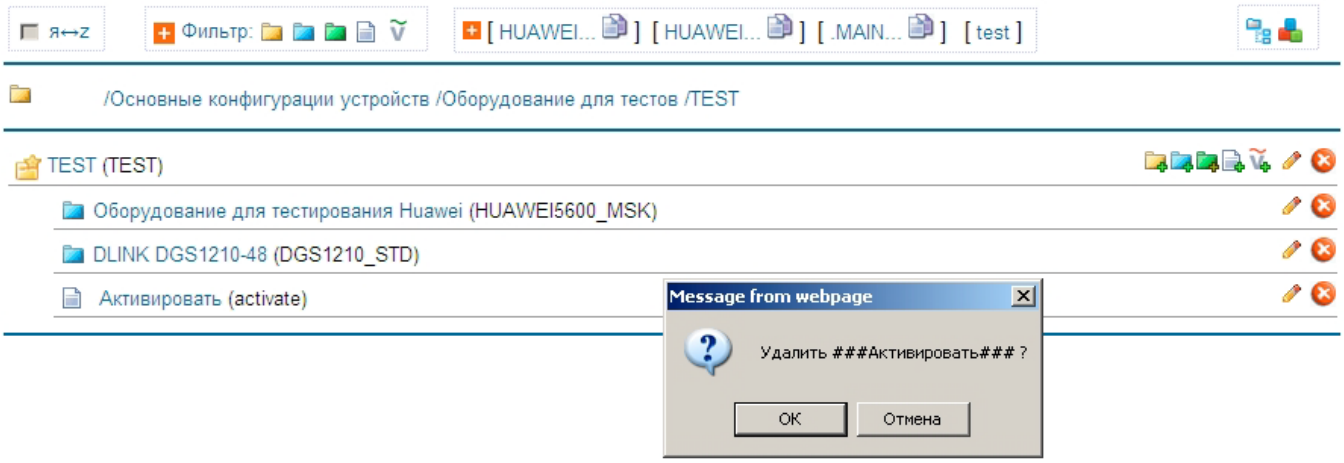

Рисунок 28 – Пример подтверждения удаления элемента

После подтверждения элемент будет удалён, что отразится в экранной форме.

4.5.4 Копирование и перенос элементов

Возможности по копированию и переносу описаны в разделах Руководства:

- 4.3.2.1 «Показать/спрятать управление»;
- 4.3.4 «Управление закладками»;
- 4.3.6 «Переключатель групп операций с элементами репозитория».

4.5.5 Редактирование элементов, общее для всех типов

Для редактирования элемента необходимо активировать, варианты:

- пиктограмму в строке выбранного элемента,
- пиктограмму самого элемента (см. п. 4.2),
- для элементов типа Сценарий и Переменная – первую часть наименования/описания (см. 4.3.2.2), выводимую ссылкой.

После активации будет открыта форма редактирования элемента.

В зависимости от типа элемента форма редактирования может иметь различный набор полей и инструментов редактирования, привязанный к набору атрибутов типа.

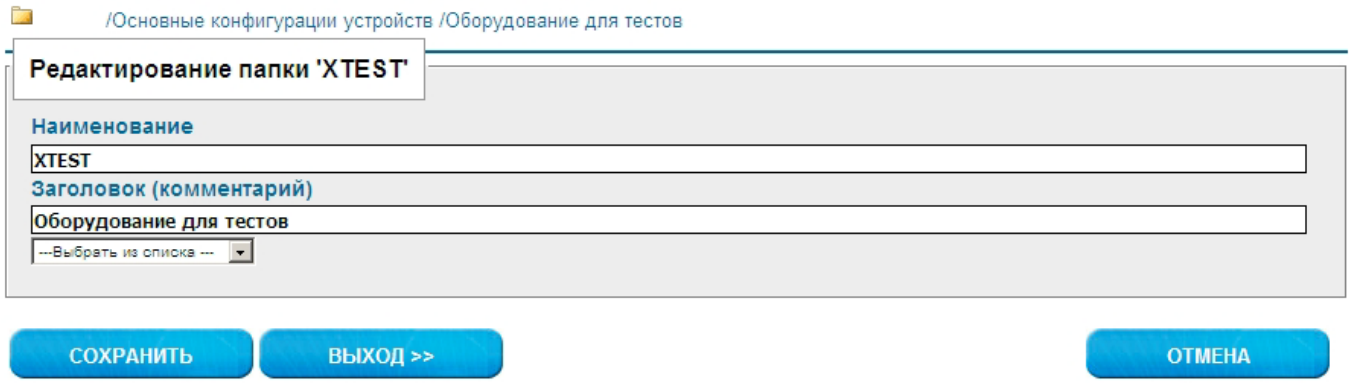

Рисунок 29 – Пример формы редактирования типа «каталог»

Наименование любого элемента должно содержать только буквы английского алфавита и цифры и начинаться с буквы.

Завершение работы с формой следует выполнить активацией одной из кнопок:

- Сохранить – внесет изменение в репозиторий без выхода из режима редактирования.
- Выход – выполнит сохранение элемента с закрытием формы редактирования.
- Отмена – закрывается форма редактирования без сохранения изменений в репозиторий.

4.5.6 Редактирование элементов типа «Каталог»

Элементы типа «Каталог» могут находиться на произвольном уровне иерархии. Требуется ввод только названия и описания (или выбор из готового списка).

Рисунок 30 – Пример редактирования каталога

4.5.7 Редактирование элементов типа «Конфигурация»

Предусмотрены три вида элементов типа «Конфигурация»:

- конфигурация модели оборудования,

- базовая конфигурация,

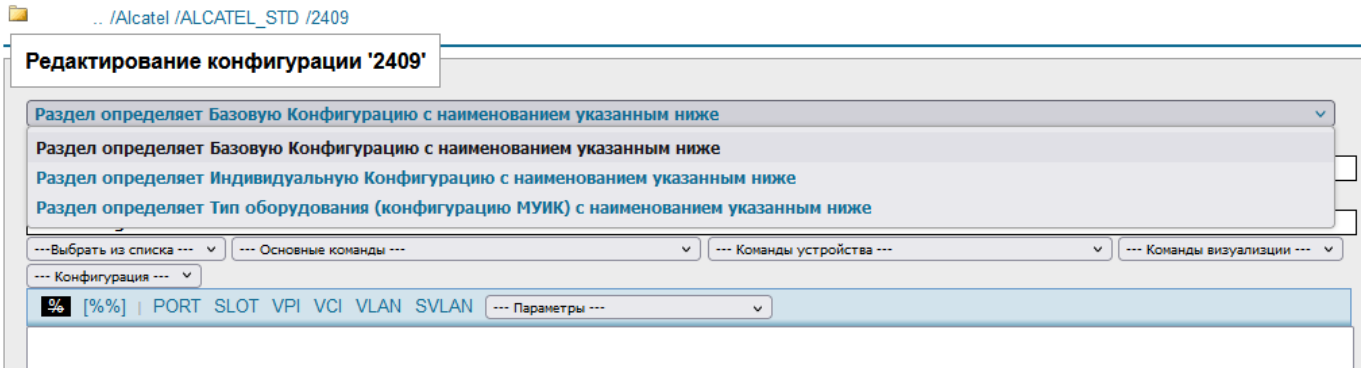

Рисунок 31 – Выбор вида конфигурации в окне редактирования

Все элементы вида «базовая конфигурация» должны находится внутри элементов первого вида. Таким образом, для каждой модели оборудования должна существовать хотя бы одна базовая конфигурация.

Рисунок 32 – Пример вложенности конфигураций

Индивидуальные конфигурации наследуются из одной базовой конфигурации.

4.5.8 Редактирование элементов типа «Параметр»

В отличие от других типов элементов, параметры могут находиться на произвольном уровне в иерархии элементов.

Для элементов типа «параметр» в дополнение к наименованию и описанию необходимо ввести два поля:

- значение по умолчанию,
- отображаемое сообщение об ошибке.

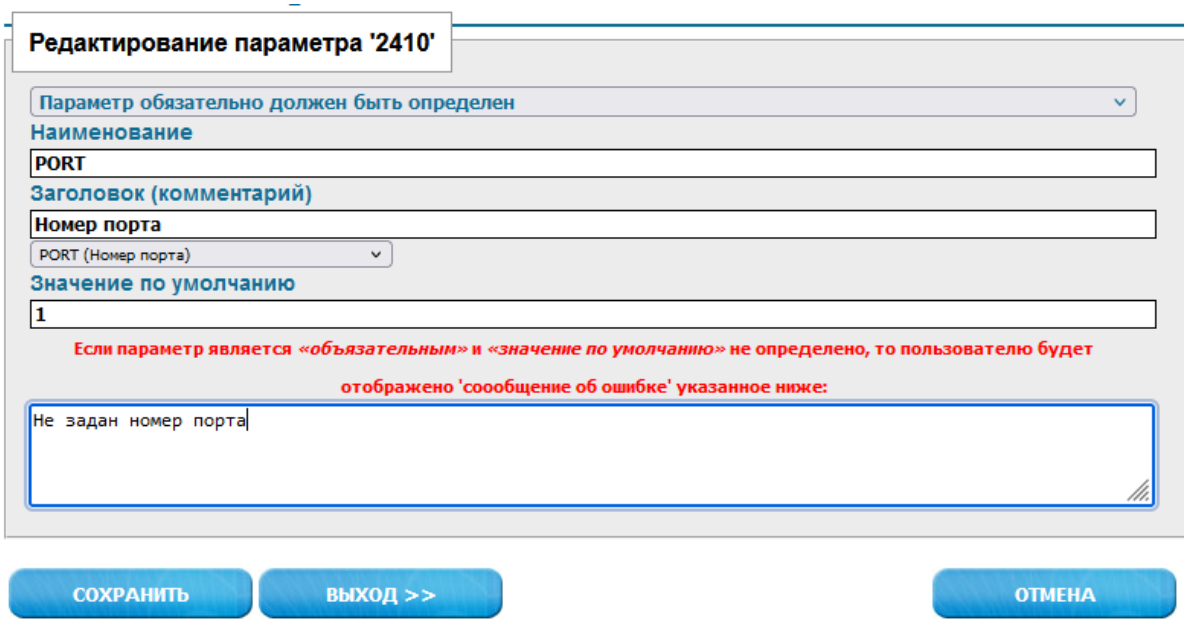

Рисунок 33 – Пример редактирования элемента типа «Параметр»

Параметры делятся на «обязательные» и «рекомендуемые».

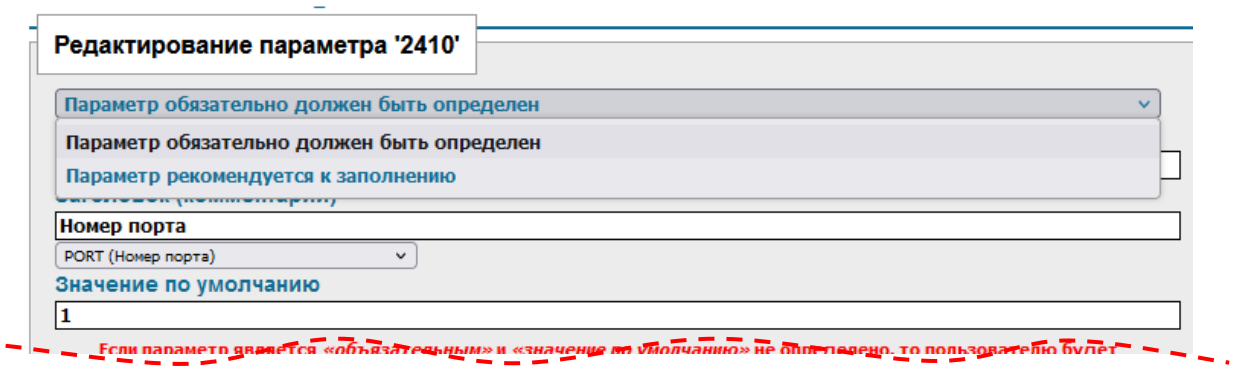

Рисунок 34 – Выбор типа параметра

4.5.9 Редактор сценариев

Редактор сценариев используется для создания и редактирования MML-команд, которые применяются при обмене с сетевыми элементами (оборудованием мультисервисного доступа) или иной целевой системой.

Подробнее о самих сценариях, логике их использования описано в соответствующих разделах Руководства.

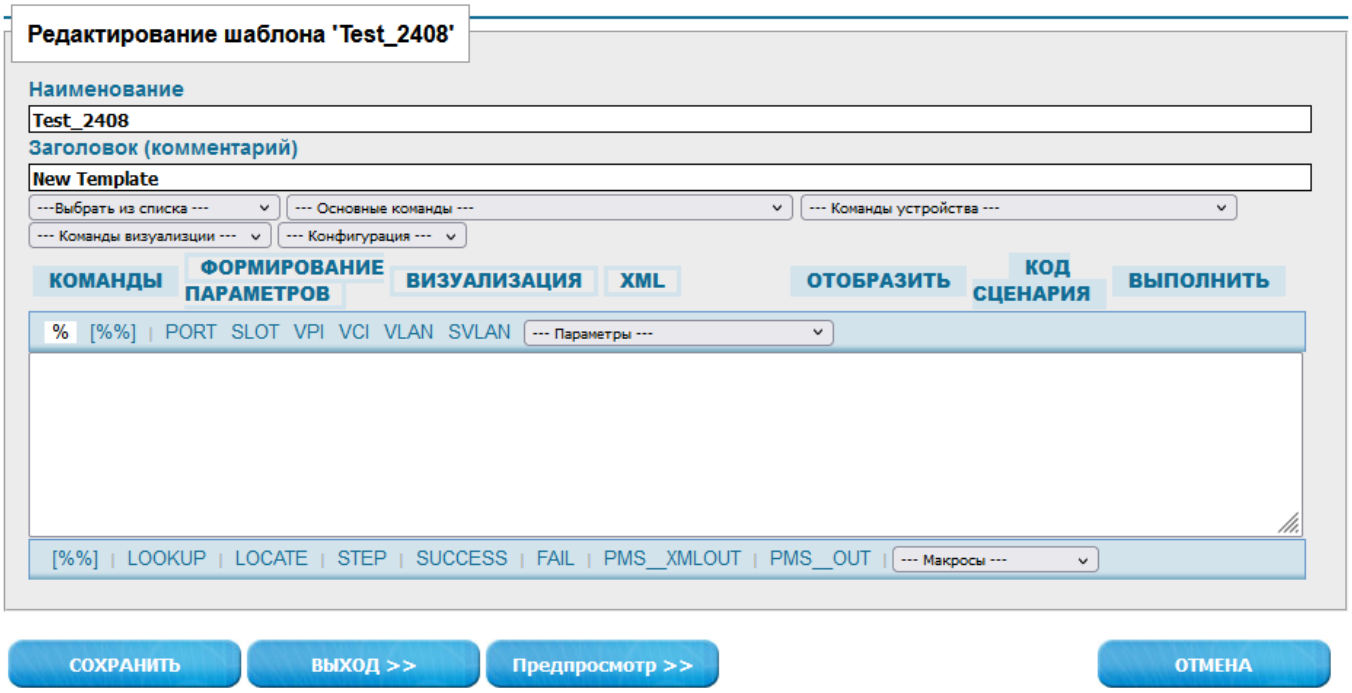

Рисунок 35 – Экранная форма «Редактор сценария»

Для удобства написания сценариев в меню редактора имеются следующие управляющие элементы:

- Элементы управления, расположенные в верхней строке – позволяют добавить в текущую позицию поля редактирования часто используемые переменные,
- Ниспадающий список, расположенный в верхней строке – позволяет добавить в текущую позицию поля редактирования переменные из справочника.
- Элементы управления, расположенные в нижней строке – позволяют добавить в текущую позицию поля редактирования часто используемые инструкции интерпретатора.
- Ниспадающий список, расположенный в нижней строке – позволяет добавить в текущую позицию поля редактирования макросы из справочника.

Далее приведено описание работы элементов управления на примерах.

В примере 1 (см. рис. 36) были последовательно активированы элементы VPI, PORT, SLOT при этом в код сценария было добавлено [%VPI%], [%PORT%], [%SLOT%]:

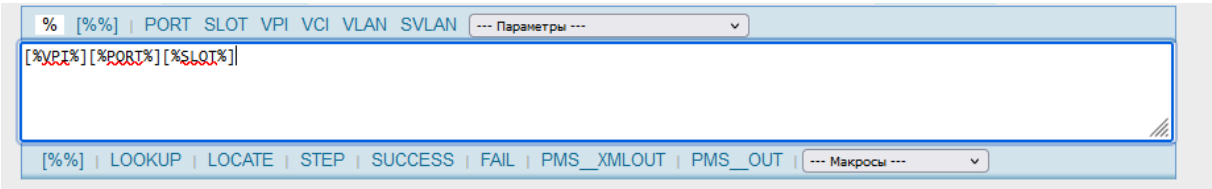

Рисунок 36 – Пример 1 редактирования сценария

В случае необходимости добавления параметров без использования [%%] необходимо переключить режим скобок («Bracket») активацией элемента ‘%’, расположенный левом углу редактора (см. рис. 36), а затем повторить действия, указанные в предыдущем примере (итог – пример 2, см. рис. 37).

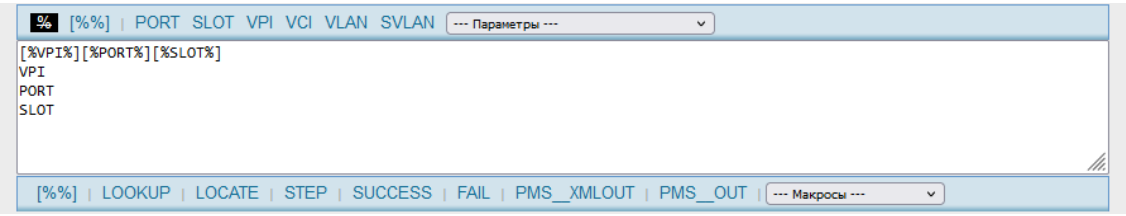

Рисунок 37 – Пример 2 редактирования сценария

Параметры из справочника вынесены в отдельный ниспадающий список. Добавление параметра в код сценария производится выбором из списка.

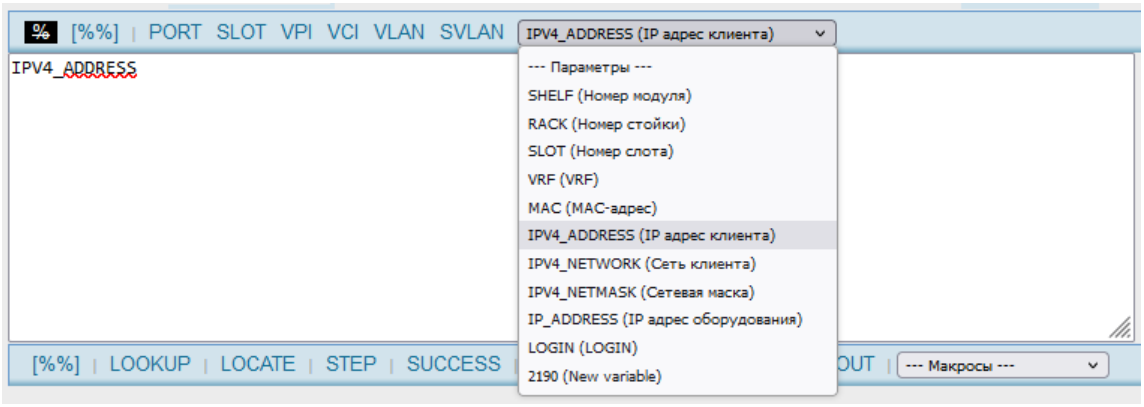

Рисунок 38 – Пример 3 редактирования сценария

Работа с элементами управления, расположенными в нижней строке редактора, производится аналогично. Пример содержимого окна редактора после активации элементов «LOOKUP», «FAIL», «STEP» изображен на следующем рисунке.

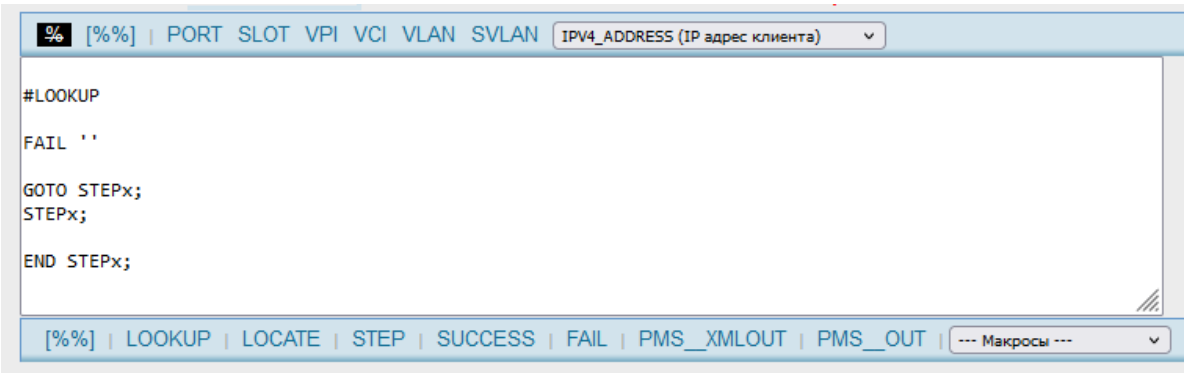

Рисунок 39 – Пример 4 редактирования сценария

Еще один вид элементов, используемых при редактировании сценария, расположен в меню «Макросы». В отличие от предыдущих элементов управления перед использованием макроса необходимо выделить часть кода в редакторе, а затем выбрать один из макросов в ниспадающем списке. К выделенному коду будет добавлено имя макроса, а сам код будет окружен скобками.

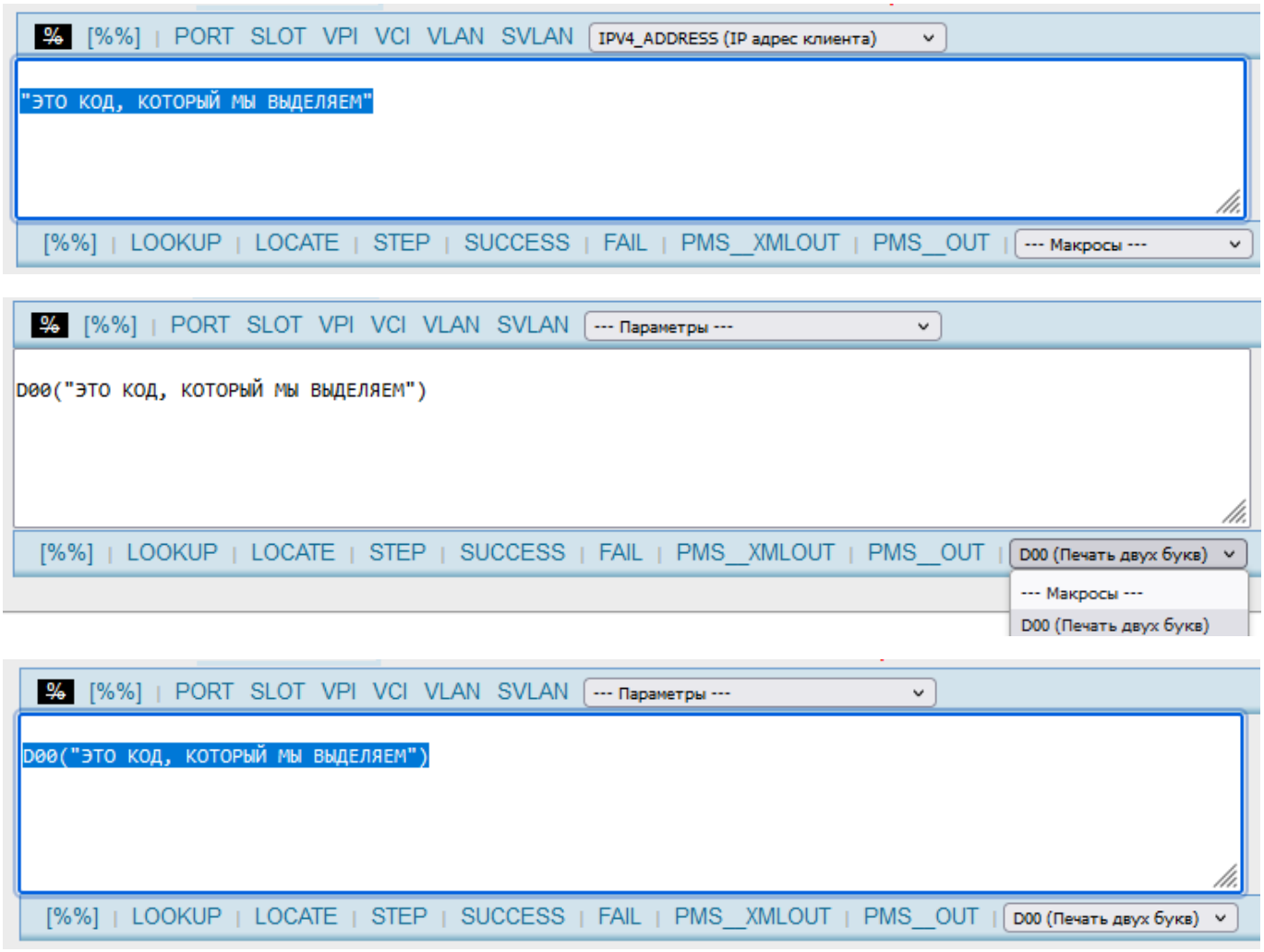

Рисунок 40 – Пример 5 редактирования сценария

4.5.10 Редактирование системного шаблона 'root'

Системный шаблон «root» используется для обработки набора технических данных, поступающих из внешней системы (см. п. 5). В шаблоне сопоставлены переменные Подсистемы и разделы набора данных, где находятся значения для них.

Рисунок 41 содержит пример редактирования системного шаблона. На нём для переменной PORT указан раздел 'Object/CommandEquipment/Equipment/Port/Name', откуда следует взять значение.

Набор активных элементов редактора системного шаблона минимален и включается в описание редактора сценариев (см. п. 4.5.9).

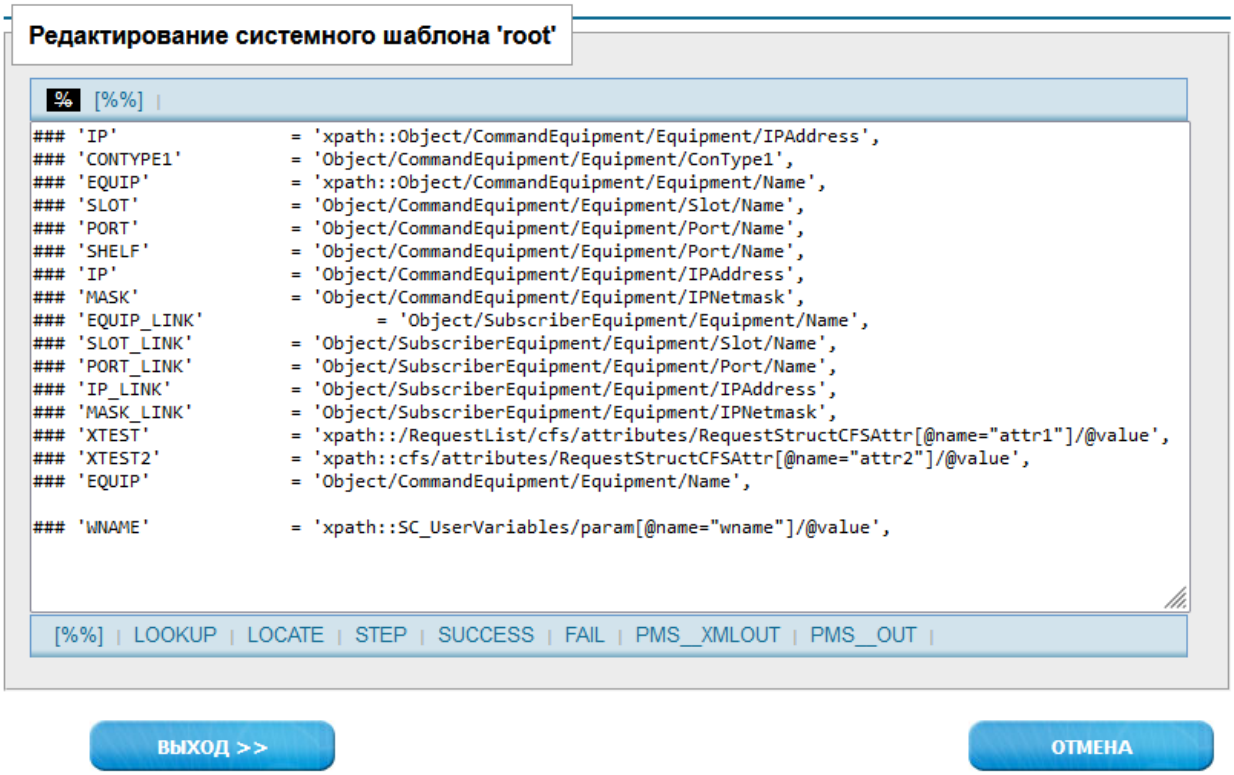

Рисунок 41 – Пример редактирования системного шаблона 'root'

5 Алгоритм поиска и подготовки сценария к выполнению

Рисунок 42 содержит схему, иллюстрирующую алгоритм поиска сценария и отправки его в модуль ШПД для выполнения на сетевом элементе.

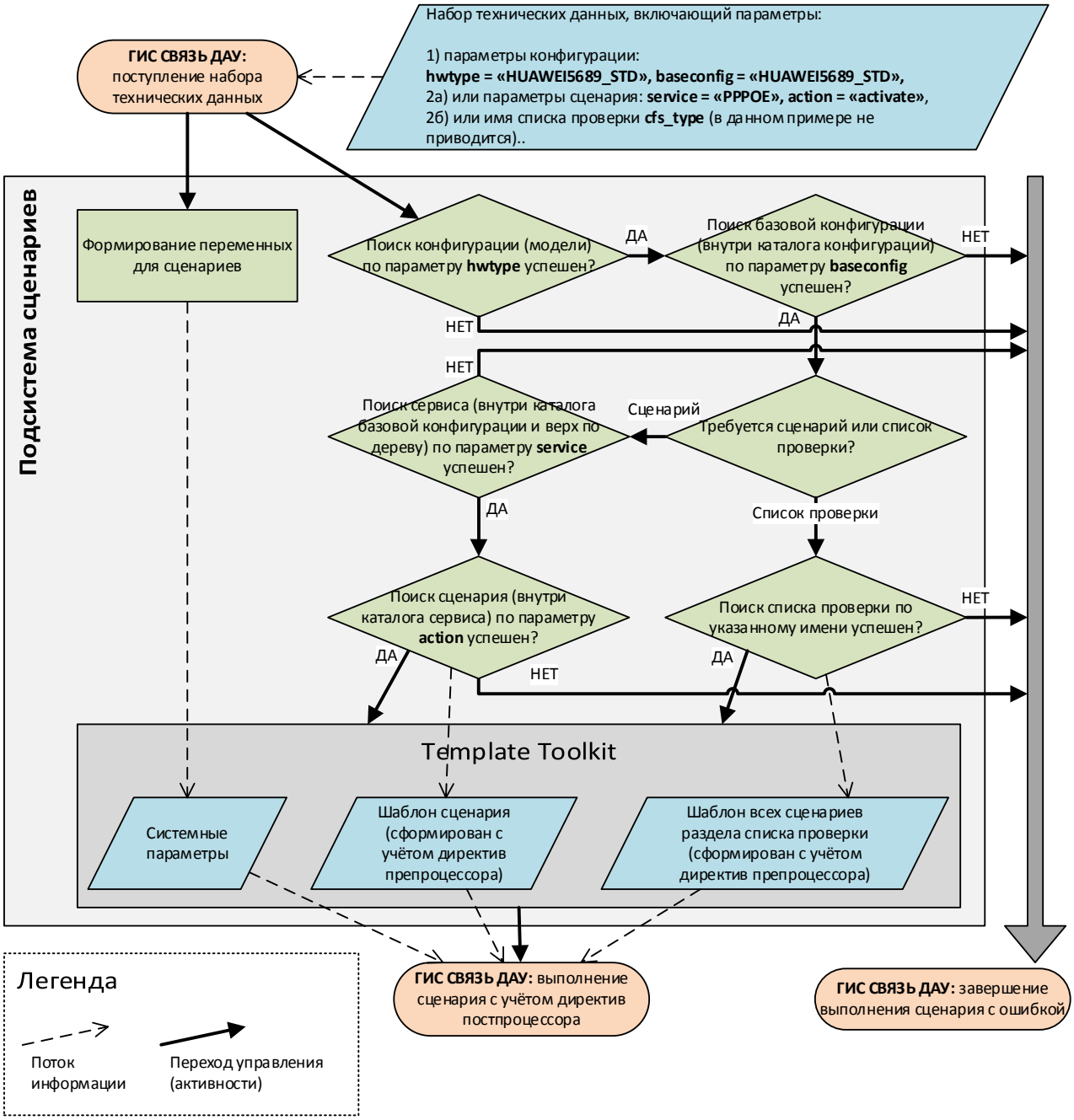

Рисунок 42 – Алгоритм поиска и подготовки сценария к выполнению с примером значений параметров

В алгоритме используются параметры из состава технических данных, которые являются необходимыми для поиска сценария. В примере на рисунке 42 параметры соответствуют элементам репозитория сценариев:

- hwtype – соответствует элементам типа «конфигурация (модели)»  ,
- baseconfig – соответствует элементам типа «базовая конфигурация»  ,
- service – соответствует элементам типа «сервис»  ,
- action – соответствует элементам типа «сценарий»  .

Описание алгоритма поиска:

1. Поиск начинается с корневого каталога репозитория. Согласно блок схеме алгоритма сначала ищется конфигурация (модель). Далее от найденного вниз по дереву ищется базовая конфигурация.
2. Принимается в работу первый найденный элемент, отвечающий критериям поиска, другие (альтернативные), далее не ищутся.
3. При поиске сервиса, в блок-схеме алгоритма фраза: «Поиск сервиса (внутри каталога базовой конфигурации и вверх по дереву)», - означает, что если необходимый сервис внутри каталога базовой конфигурации не найден, то поиск начинается по дереву «снизу-вверх» от найденной базовой конфигурации вплоть до корневой папки. Таким образом, можно описывать один сервис на несколько базовых конфигураций и даже моделей.
4. При поиске списка проверок:
   a. В искомый список включаются все сценарии, находящиеся непосредственно в папке найденного списка проверок.
   b. Если найденная папка списка проверок не содержит ни одного сценария, то возвращается пустой список.
   c. Если в папке базовой конфигурации список проверок с указанным именем не найден, то поиск начинается по дереву «снизу-вверх» от найденной базовой конфигурации вплоть до корневой папки. Таким образом, можно описывать один список проверок на несколько базовых конфигураций и даже моделей.

6 Проверки сервисов

6.1 Элемент Проверки, общие сведения

Элемент «Список проверок» (кратко – Проверки) предназначен для размещения набора сценариев проверки сервисов. Элемент типа «Список проверок» может содержать только подчиненные элементы типа «Сценарий».

Различия между типами элементов «сервис» и Проверки описаны в п. 4.2.

Разделы, относящиеся к типу «Список проверок», обозначены графическим знаком .

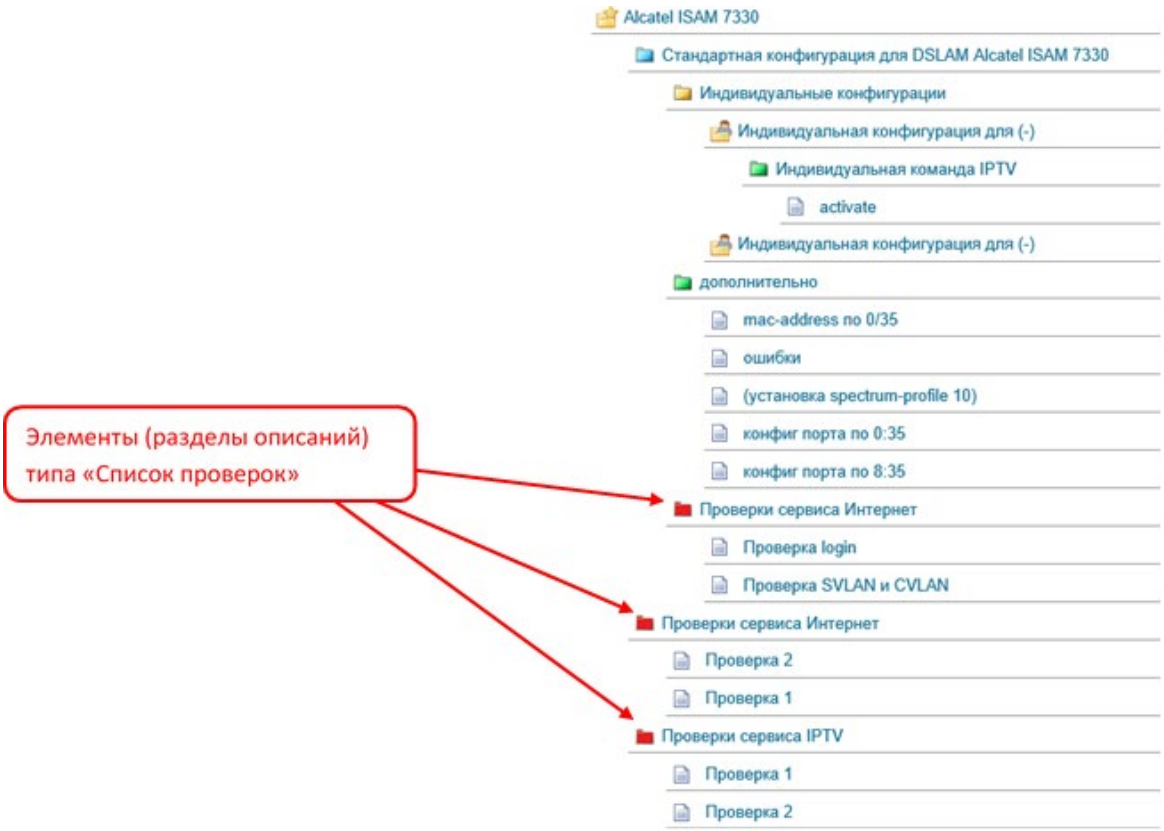

Рисунок 43 – Пример размещения в репозитории разделов типа «Список проверок»

Для использования элемента реализовано два варианта создания проверок сервисов:

- Первый вариант: создание сценариев «вручную», описано в п. 6.2.».
- Второй вариант: автоматизированное создание сценария (создание с использованием «Мастера создания сценариев»), описано в п. 6.3.

Для написания сценариев проверки сервисов предусмотрены специальные директивы постпроцессора, описание которых приведены в п. 8.3.

Примеры по проверкам сервисов приведены в Приложениях 1 – 3.

6.2 Создание списков проверок «вручную»

Для создания раздела типа «Список проверок» в предусмотрен элемент управления «Создать список проверок». Он доступен в следующих разделах главного меню: Конфигурации, Скрипты, Справочники, Сервисы, Действия, Параметры, Макросы.

Элемент управления «Создать список проверок» в экранных формах выглядит следующим образом:

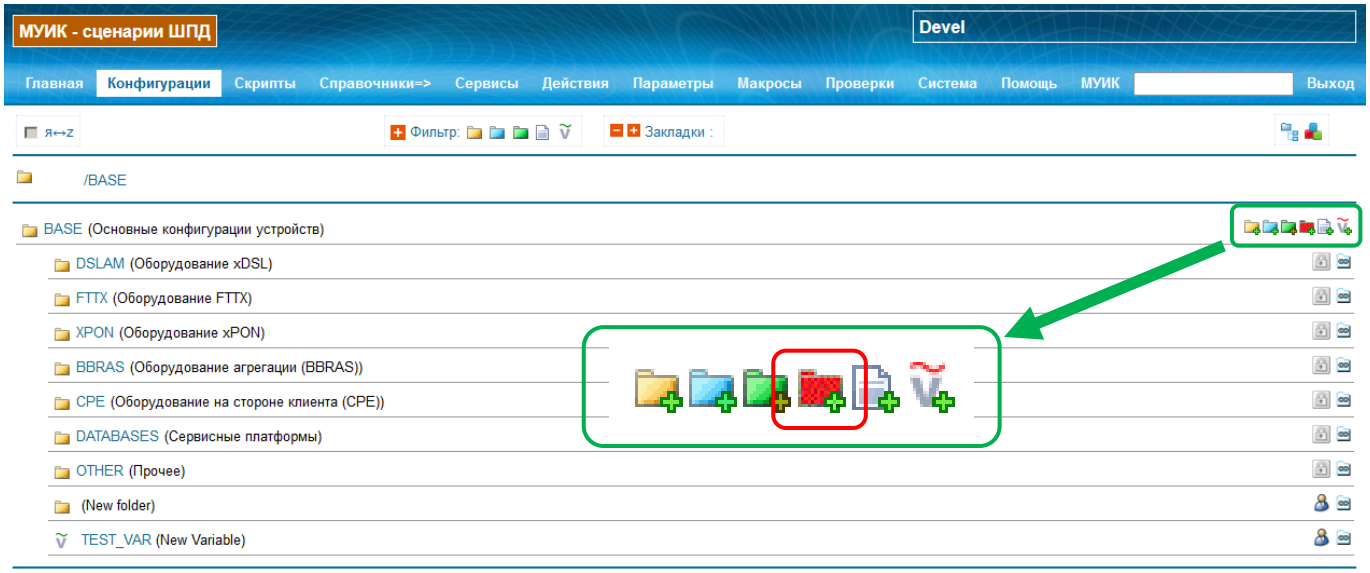

Рисунок 44 – Обозначения элемента управления «Создать список проверок» и пример его размещения в экранной форме

Раздел типа «Список проверок» следует размещать на том же уровне, где размещаются элементы типа «сервис» в базовых конфигурациях оборудования.

При создании и редактировании списка сценариев следует учесть, что сценарии выполняются в той последовательности, в которой они выводятся в составе экранной формы, сверху вниз.

6.3 Автоматизация процесса создания сценариев пользователем

6.3.1 Открытие и общий вид экранной формы «Мастер создания сценариев»

Мастер создания сценариев (Мастер) расположен в разделе Проверки. Общий вид экранной формы Мастера:

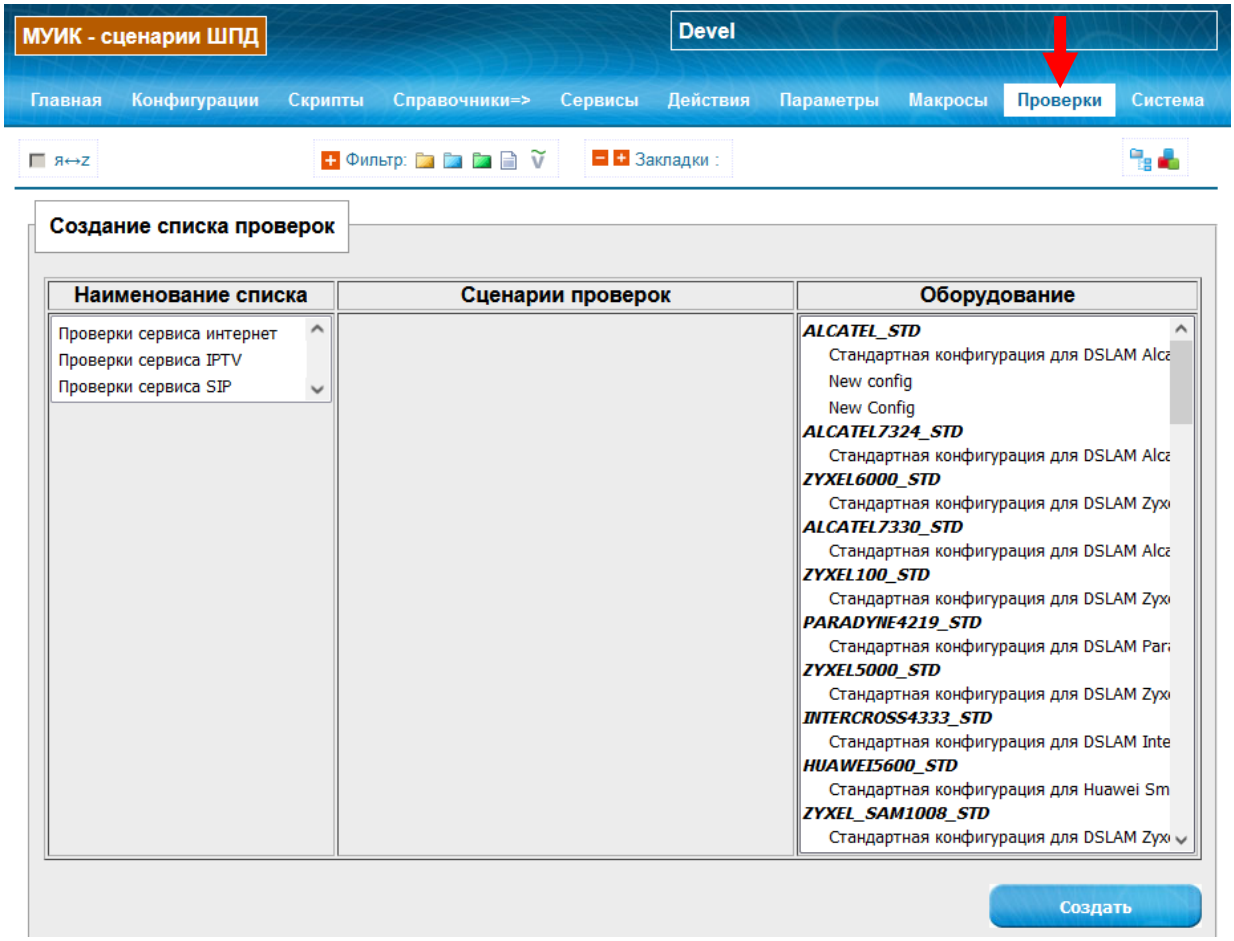

Рисунок 45 – Стартовый вид «Мастера создания сценариев» (экранная форма «Создание списка проверок»)

6.3.2 Порядок работы с «Мастером создания сценариев»

Порядок создания элемента списка проверок:

1. В колонке «Наименование списка» следует выбрать наименование списка сценариев проверок из справочника. Описание справочника проверок приведено в п. 6.4.
2. По выбранному наименованию в колонке «Сценарии проверок» появится соответствующий список проверок.
3. Выбрать из списка один или несколько сценариев проверки. Доступен множественный выбор (использовать удерживание клавиши «Ctrl»).
4. Выбрать одну или несколько базовых конфигураций.

5. Кнопкой «Создать» инициировать генерацию нового списка проверок.
6. В поле информации появится список новых проверок со ссылками на место расположения.

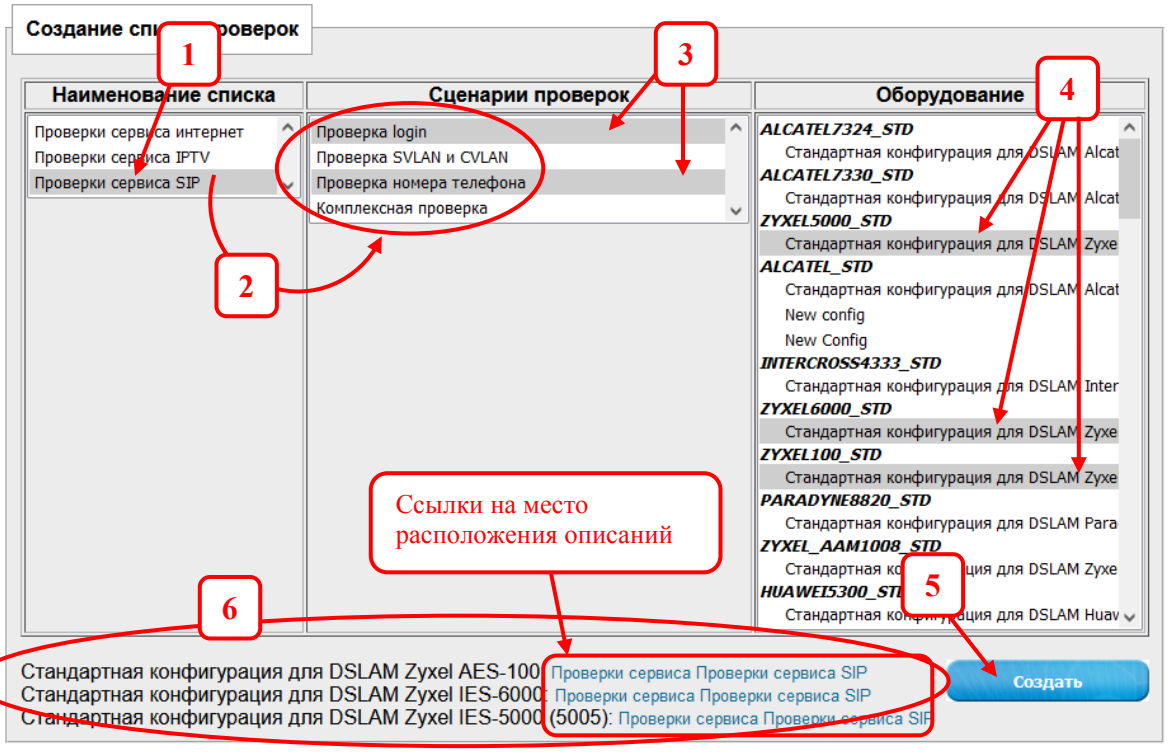

Рисунок 46 – Последовательность работы в экранной форме «Создание списка проверок»

6.4 Место размещения и администрирование справочника списков проверок

Свойства справочников списков проверок:

1. Справочник «Списки сценариев проверок» (в форме Мастера – колонка «Наименование списка»).

   Пример размещения списков указано на иллюстрации Рисунок 47.

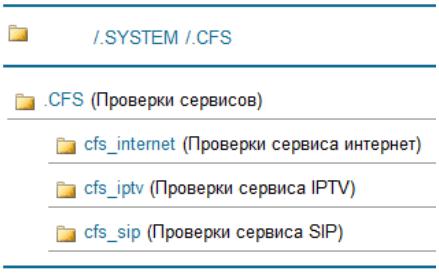

Рисунок 47 – Размещение списков сценариев проверок

Наполнение списка возможных проверок производится на основании сервисной модели, принятой в организации, где развёрнут экземпляр ПО.

2. Справочник «Сценарии проверок» (в форме колонка «Сценарии проверок»).

Списки сценариев проверок могут содержать предопределённый набор сценариев-шаблонов, которые будут копированы в соответствующие (выбранные в форме Мастера создания сценариев) базовые конфигурации оборудования.

Размещение сценариев проверок проиллюстрировано на рисунке 48.

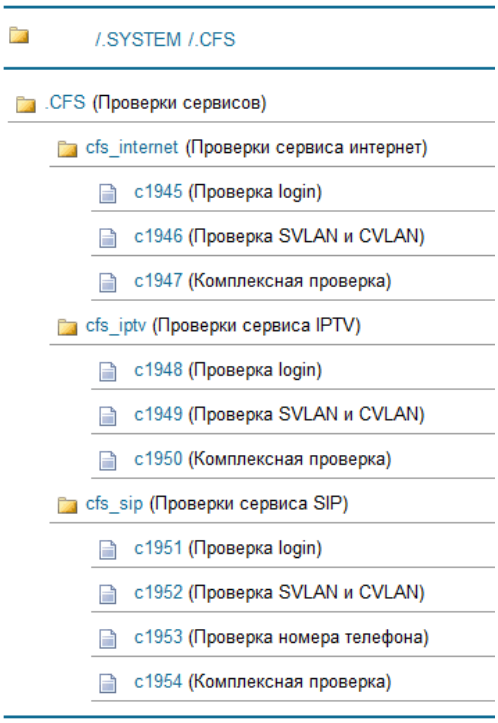

Рисунок 48 – Размещение сценариев проверок в списках

Администрирование содержимого разделов (набора сценариев), как и каждого из сценариев, выполняется пользователем с ролью Администратор ШПД.

7 Препроцессор

7.1 Состав сценариев

Сценарии состоят из:

- MML-инструкций или строк-запросов, которые будут отправлены на сетевой элемент;
- инструкций препроцессора и кода шаблонизатора;
- инструкций постпроцессора.

Пример MML-инструкции:

```
show version
```

Пример строки-запроса:

```
@http get http://domen.com/search?string=query
```

Пример инструкции препроцессора:

```
#LOCATE functions.tpl /FUNCTIONS.dir
```

Пример кода шаблонизатора:

```
[%SLOT = (SLOT.length) ? SLOT : SLOT_LINK%]
```

Пример MML-инструкции, включающей код шаблонизатора:

```
info configure bridge port 1/1/[%SLOT%]/[%PORT%]:0:35 detail
```

Пример инструкции постпроцессора:

```
GOTO STEP1
```

7.2 Препроцессор (шаблонизатор)

Настоящий раздел содержит описание инструкции препроцессора и шаблонизатора. В качестве шаблонизатора используется Template Toolkit, основанный на PERL (см. http://template-toolkit.org).

7.2.1 Синтаксис

Раздел описывает синтаксис, структуру и семантику директив и языка общего представления Template Toolkit.

Директивы шаблонизатора заключены между последовательностями символов '[%' и '%]'. Например,

```
[% PROCESS header %]
```

Директивы допускается включать в любом месте текста и разбивать на несколько строк. Дополнительные пробельные символы внутри директивы в общем случае несущественны.

```
[% INCLUDE header
title = 'Hello World'
bgcol = '#ffffff'
%]
[% INCLUDE menu align='right'%]
Name: [% name %]  ([%id%])
```

Комментарии

Символ '#' используется для обозначения комментария внутри директивы. Если вставить символ сразу за открывающим тегом вся директива целиком будет проигнорирована.

```
[%# вся директива целиком игнорируется
вне зависимости от того на сколько строк она разбита
%]
```

В любом другом месте этот символ указывает, что оставшаяся часть строки является комментарием.

```
[% # это комментарий
theta = 20      # so is this
rho   = 30      # <aol>me too!</aol>
%]
```

Блочные директивы

Такие директивы как FOREACH, WHILE, BLOCK и т.д. обозначают начало блока, который может содержать текст или другие директивы и который завершается соответствующей директивой END. Вложенность блоков не ограничена. Директивы IF, UNLESS, ELSIF и ELSE также определяют блоки и могут использоваться друг с другом по обычным правилам.

```
[% FOREACH item = [ 'foo' 'bar' 'baz' ] %]
 * Item: [% item %]
[% END %]

[% BLOCK footer %]
 Copyright 2000 [% me %]
 [% INCLUDE company/logo %]
[% END %]
```

```
[% IF foo %]
 [% FOREACH thing = foo.things %]
  [% thing %]
 [% END %]
[% ELSIF bar %]
 [% INCLUDE barinfo %]
[% ELSE %]
 do nothing...
[% END %]
```

Блочные директивы также могут использоваться в удобной обратной нотации.

```
[% INCLUDE userinfo FOREACH user = userlist %]
[% INCLUDE debugtxt msg="file: $error.info"
 IF debugging %]
[% "Danger Will Robinson" IF atrisk %]
```

Блоки директив

Несколько директив, разделенных символом ';', могут быть заключены между одной парой тегов.

```
[% IF title;
 INCLUDE header;
ELSE;
 INCLUDE other/header  title="Some Other Title";
END
%]
```

7.2.2 Директивы

Раздел содержит справку о директивах Template Toolkit с примерами использования.

7.2.2.1 Получение и обновление переменных шаблона

GET

Директива GET получает и выводит значение указанной переменной.

```
[% GET foo %]
```

Ключевое слово GET можно опустить, достаточно указать имя переменной внутри тегов директивы.

```
[% foo %]
```

Переменная может иметь неограниченное число элементов, в качестве разделителя между ними используется знак «.» точка. Каждому элементу можно передавать аргументы, указав их в круглых скобках.

```
[% foo %]
[% bar.baz %]
[% biz.baz(10) %]
... и т.д. ...
```

Более подробное описание переменных шаблона приведено в п. 7.2.3.

Допускается использовать выражения, построенные с помощью логических (and, or, not, ?:) и математических (+ - * / % mod div) операторов.

```
[% template.title or default.title %]
[% score * 100 %]
[% order.nitems ? checkout(order.total) : 'no items' %]
```

Оператор 'div' возвращает целую часть результата деления. Оба оператора '%' и 'mod' возвращают остаток от деления. 'mod' является синонимом для '%'

```
[% 15 / 6 %]  # 2.5
[% 15 div 6 %]  # 2
[% 15 mod 6 %]  # 3
```

SET

Директива SET позволяет присваивать существующим переменным новые значения или создавать новые временные переменные.

```
[% SET title = 'Hello World' %]
```

Ключевое слово SET можно опустить.

```
[% title = 'Hello World' %]
```

Переменным можно присваивать значения других переменных, числа, 'текст в одиночных кавычках' (литералы) и "текст в двойных кавычках". В последнем случае любая ссылка на переменную внутри текста будет интерполироваться во время вычисления строки. Переменные должны начинаться с символа '$' и могут выделяться фигурными скобками для явного выделения имени переменной там, где это необходимо.

```
[% foo  = 'Foo'  %]   # литерал 'Foo'
[% bar  =  foo   %]   # значение переменной 'foo'
[% cost = '$100' %]   # литерал '$100'
```

```
[% item = "$bar: ${cost}.00" %]    # значение интерполяции строки "Foo: $100.00"
```

Внутри одной директивы можно выполнить несколько присваиваний в порядке их указания. Так, приведенный выше пример можно переписать следующим образом:

```
[% foo  = 'Foo'
bar  = foo
cost = '$100'
item = "$bar: ${cost}.00"
%]
```

Как и в GET можно использовать простые выражения.

```
[% ten    = 10
twenty = 20
thirty = twenty + ten
forty  = 2 * twenty
fifty  = 100 div 2
six    = twenty mod 7
%]
```

Конкатенция

Конкатенция (склеивание) строк выполняется с помощью оператора ' _ '. Оператор должен указываться с пробелами с обеих сторон:

```
[% copyright = '(C) Copyright' _ year _ ' ' _ author %]
```

Вариант - использовать интерполяцию текста в двойных кавычках:

```
[% copyright = "(C) Copyright $year $author" %]
```

7.2.2.2 Обработка других блоков

INCLUDE

Директива INCLUDE используется для обработки и включения выходного потока другого шаблона или блока.

```
[% INCLUDE header %]
```

После имени шаблона можно указать локальные переменные, которые временно скрывают (маскируют) все существующие переменные с такими же именами. Лишние пробельные символы внутри директивы игнорируются, поэтому можно добавить определения переменных на той же строке, следующей строке или разбить на несколько строк, сопроводив их комментариями, если это предпочтительней.

```
[% INCLUDE table %]
[% INCLUDE table title="Active Projects" %]
[% INCLUDE table
title   = "Active Projects"
bgcolor = "#80ff00"     # бледно-зеленный
border  = 2
%]
```

Директива INCLUDE локализует (т.е. создает локальные копии) всех переменных перед обработкой шаблона. Все изменения внутри включаемого шаблона не окажут влияния на переменные в родительском шаблоне.

```
[% foo = 10 %]
foo изначально [% foo %]
[% INCLUDE bar %]
foo опять [% foo %]
[% BLOCK bar %]
 foo было [% foo %]
 [% foo = 20 %]
 foo стало [% foo %]
[% END %]
```

Результат, полученный на выходе:

```
foo изначально 10
foo было 10
foo стало 20
foo опять 10
```

Если требуется обработать несколько шаблонов за один проход, можно указать через '+' их имена (в кавычках или без, переменные только в двойных кавычках). Директива INCLUDE обработает их по порядку.

```
[% INCLUDE html/header + "site/$header" + site/menu
title = "My Groovy Web Site"
%]
```

PROCESS

Директива PROCESS используется для обработки и включения выходного потока другого блока. Директива не выполняет локализации переменных перед обработкой шаблона, т.е. любые изменения, сделанные в переменных внутри включаемого шаблона отразятся в родительском шаблоне.

```
[% foo = 10 %]
foo - [% foo %]
[% PROCESS bar %]
foo - [% foo %]
[% BLOCK bar %]
 [% foo = 20 %]
 меняем foo на [% foo %]
[% END %]
```

На выходе:

```
foo - 10
меняем foo на 20
foo - 20
```

Нет необходимости заключать в кавычки первый параметр, если он содержит только латинские символы, цифры, подчеркивания, точки и прямые слэши. Префикс '$' можно использовать, чтобы явно указать, что имя шаблона следует получить из переменной:

```
[% myheader = 'my/misc/header' %]
[% PROCESS  myheader %]  # 'myheader'
[% PROCESS $myheader %]  # 'my/misc/header'
```

Можно указать несколько шаблонов, разделенных '+', и они будут обработаны по порядку.

```
[% PROCESS html/header + my/header %]
```

BLOCK

Конструкция BLOCK ... END может быть использована для определения шаблона компонента, который может быть обработан директивой PROCESS.

```
[% BLOCK tabrow %]
 <tr><td>[% name %]</td><td>[% email %]</td></tr>
[% END %]
<table>
[% PROCESS tabrow  name='Fred'  email='fred@nowhere.com' %]
[% PROCESS tabrow  name='Alan'  email='alan@nowhere.com' %]
</table>
```

Определение BLOCK можно использовать до того, как оно будет явно определенно. Само определение блока не генерирует никакого вывода.

```
[% PROCESS tmpblk %]
[% BLOCK tmpblk %] This is OK [% END %]
```

Анонимный BLOCK можно использовать для захвата вывода фрагмента шаблона.

```
[% julius = BLOCK %]
 And Caesar's spirit, ranging for revenge,
 With Ate by his side come hot from hell,
 Shall in these confines with a monarch's voice
 Cry  'Havoc', and let slip the dogs of war;

 That this foul deed shall smell above the earth

 With carrion men, groaning for burial.
[% END %]
```

Как и именованный блок, он может содержать любые директивы шаблона, которые обрабатываются в момент определения блока. Вывод генерируется в момент присвоения переменной 'julius'.

7.2.2.3 Условная обработка

IF / UNLESS / ELSIF / ELSE

Директивы IF и UNLESS позволяют обрабатывать (или пропускать) блоки на основании выполнения некоторого условия.

```
[% IF frames %]
 [% PROCESS frameset %]
[% END %]
[% UNLESS text_mode %]
 [% PROCESS biglogo %]
[% END %]
```

Несколько условий можно связать с помощью блоков ELSIF и/или ELSE.

```
[% IF age < 10 %]
 Hello [% name %], does your mother know you're using her AOL account?
[% ELSIF age < 18 %]
 Sorry, you're not old enough to enter (and too dumb to lie about your age)
[% ELSE %]
 Welcome [% name %].
[% END %]
```

Допускается использовать следующие условные и логические операторы:

```
== != < <= > >= && || ! and or not
```

Операторы 'and', 'or' и 'not' являются синонимами для '&&', '||' и '!', соответственно.

Условия могут быть составными. Для точного определения порядка вычисления используются скобки.

```
# надуманный пример составного условия

[% IF (name == 'admin' || uid <= 0) && mode == 'debug' %]
 I'm confused.
[% ELSIF more > less %]
 That's more or less correct.
[% END %]
```

SWITCH / CASE

Конструкцию SWITCH / CASE можно использовать для выбора из множества вариантов. В директиве SWITCH указывается выражение, которое вначале вычисляется, а затем результат сравнивается по очереди с выражениями в CASE. Каждая директива CASE может содержать одно или список выражений, с которыми будет проводиться сравнение. Также можно оставить CASE пустой, либо записать [% CASE DEFAULT %] для того, чтобы определить выбор по умолчанию. Обрабатывается только одно совпадение CASE. Сквозной проход через все выражения CASE не поддерживается.

```
[% SWITCH myvar %]
 [% CASE value1 %]
 ...
 [% CASE [ value2 value3 ] %]   # множественные значения
 ...
 [% CASE myhash.keys %]         # то же самое
 ...
 [% CASE %]                     # по умолчанию
 ...
[% END %]
```

7.2.2.4 Циклическая обработка

FOREACH

Директива FOREACH циклически перебирает все значения массива, обрабатывая блок для каждого из них.

```
[% foo = 'Foo'
items = [ 'one', 'two', 'three' ] %]
```

Шаблон:

```
Things:
[% FOREACH thing = [ foo 'Bar' "$foo Baz" ] %]
 * [% thing %]
[% END %]

Items:
[% FOREACH i = items %]
 * [% i %]
[% END %]

Stuff:
[% stuff = [ foo "$foo Bar" ] %]
[% FOREACH s = stuff %]
 * [% s %]
[% END %]
```

Вывод:

```
Things:
* Foo
* Bar
* Foo Baz

Items:
* one
* two
* three

Stuff:
* Foo
* Foo Bar
```

Вместо '=' можно использовать 'IN'.

```
[% FOREACH crook IN government %]
```

Директива **NEXT** начинает следующий проход цикла FOREACH.

```
[% FOREACH user IN userlist %]

 [% NEXT IF user.isguest %]

 Name: [% user.name %]    Email: [% user.email %]
```

```
[% END %]
```

Директива **LAST** используется для преждевременного завершения цикла. **BREAK** используется в качестве синонима LAST.

```
[% FOREACH match IN results.nsort('score').reverse %]
 [% LAST IF match.score $lt; 50 %]
 [% match.score %] : [% match.url %]
[% END %]
```

WHILE

Директиву WHILE можно использовать для повторного выполнения блока-шаблона до тех пор, пока выполняется (т.е. истинно) условное выражение. Как и в IF / UNLESS, выражение может быть составным.

```
[% WHILE total < 100 %]
 ...
 [% total = calculate_new_total %]
[% END %]
```

Так же как в FOREACH, для начала следующей итерации можно использовать директиву NEXT, а для завершения цикла - BREAK.

Template Toolkit использует надежный счетчик для того чтобы избежать зацикливания циклов WHILE. Если цикл превысит 1000 итераций, будет сгенерировано исключение со следующим сообщением об ошибке:

```
WHILE loop terminated (> 1000 iterations)
```

7.2.2.5 Плагины и макросы

Плагин - это обычный модуль Perl, соответсвующий специальному объектно-ориентированному интерфейсу, позволяющему Template Toolkit автоматически его загружать и использовать.

Имена плагинов чувствительны к регистру.

Некоторые стандартные плагины включены в пакет Template Toolkit.

USE

Директива USE используется для загрузки и инициализации модулей расширения (плагинов).

```
[% USE myplugin %]
```

MACRO

Директива MACRO позволяет определить директиву или блок директив, которые будут вычисляться при вызове макроса.

```
[% MACRO header INCLUDE header %]
```

После этого вызов макроса:

```
[% header %]
```

будет эквивалентен:

```
[% INCLUDE header %]
```

Макросам можно передавать при вызове именованные параметры. Значения этих параметров будут локальными внутри макроса.

```
[% header(title='Hello World') %]
```

эквивалентно:

```
[% INCLUDE header title='Hello World' %]
```

В определении MACRO можно указать имена параметров. Значения, передаваемые макросу, затем связываются с этими локальными переменными. Следом можно указать другие именованные параметры.

```
[% MACRO header(title) INCLUDE header %]
[% header('Hello World') %]
[% header('Hello World', bgcol='#123456') %]
```

эквивалентно:

```
[% INCLUDE header title='Hello World' %]

[% INCLUDE header title='Hello World' bgcol='#123456' %]
```

Ниже другой пример определения макроса для вывода номеров группами по 3 цифры, разделенных запятой, с использованием виртуальных методов chunk и join.

```
[% MACRO number(n) GET n.chunk(-3).join(',') %]
[% number(1234567) %]  # 1,234,567
```

Можно определить MACRO как анонимный BLOCK. Этот блок будет вычисляться каждый раз при вызове макроса.

```
[% MACRO header BLOCK %]
 ...content...
```

```
[% END %]
[% header %]
```

7.2.2.6 Обработка исключений

TRY / THROW / CATCH / FINAL

Template Toolkit поддерживает полнофункциональную, многоуровневую обработку исключений. Директива TRY открывает блок обработки исключений, который продолжается до соответствующей директивы END. Любые ошибки, произошедшие внутри этого блока, будут перехвачены, и могут быть обработаны одним из определенных блоков CATCH.

```
[% TRY %]
 ...blah...blah...
 [% CALL somecode %]
 ...etc...
 [% INCLUDE someblock %]
 ...and so on...
[% CATCH %]
 An error occurred!
[% END %]
```

Ошибки возбуждаются как исключения и содержат два поля: 'type' и 'info'. Исключение 'type' может быть любой строкой, содержащей латинские буквы, числа, '_' или '.', и используется для определения типа произошедшей ошибки. Поле 'info' содержит сообщение об ошибке, обозначающее, что на самом деле произошло не так. Внутри блока-перехватчика (CATCH) объект исключения доступен как переменная 'error'. Допускается обращение к полям 'type' и 'info' напрямую.

```
[% mydsn = 'dbi:MySQL:foobar' %]
...
[% TRY %]
 [% USE DBI(mydsn) %]

[% CATCH %]

 ERROR! Type: [% error.type %]
 Info: [% error.info %]
[% END %]
```

Вывод (предполагается, что несуществующая база данных называется 'foobar'):

```
ERROR!  Type: DBI
Info: Unknown database "foobar"
```

Также можно использовать саму переменную 'error' и на выходе получить строку вида "$type error - $info".

```
 ...
[% CATCH %]
 ERROR: [% error %]
[% END %]
```

Вывод:

```
ERROR: DBI error - Unknown database "foobar"
```

Каждому блоку CATCH можно указать определенный тип исключения, который он будет обрабатывать. Для обработки различных типов исключений, которые могут быть возбуждены в блоке TRY, можно использовать несколько блоков CATCH. Блок CATCH без указания типа, как в предыдущем примере, является обработчиком по умолчанию и будет перехватывать все непе-рехваченные другими блоками исключения. Можно также определить этот блок как [% CATCH DEFAULT %].

```
[% TRY %]
 [% INCLUDE myfile %]
 [% USE DBI(mydsn) %]
 [% CALL somecode %]
 ...
[% CATCH file %]
 File Error! [% error.info %]
[% CATCH DBI %]
 [% INCLUDE database/error.html %]
[% CATCH %]
 [% error %]
[% END %]
```

Допускается записывать несколько директив внутри одной пары тегов, разделяя их ';'. Таким образом, простые блоки CATCH можно записать более кратко:

```
[% TRY %]
 ...
[% CATCH file; "File Error! $error.info" %]
[% CATCH DBI;  INCLUDE database/error.html %]
[% CATCH; error %]
[% END %]
```

или даже:

```
[% TRY %]
 ...
[% CATCH file ;
```

```
 "File Error! $error.info" ;
CATCH DBI ;
 INCLUDE database/error.html ;
CATCH ;
 error ;
END
%]
```

Ошибка 'file' автоматически генерируется Template Toolkit, когда он не может найти, загрузить, разобрать или обработать шаблон, который был запрошен через директивы INCLUDE и PROCESS. Если в приведенном выше примере 'myfile' не может быть найден, директива [% INCLUDE myfile %] возбудит исключение 'file', которое затем перехватывается блоком [% CATCH file %], генерируя следующий вывод:

```
File Error! myfile: not found
```

Не перехваченные исключения (например, если блок TRY не имеет перехватчика этого типа или перехватчика по умолчанию) могут быть обработаны окружающими блоками TRY, которые могут иметь неограниченную вложенность в пределах нескольких шаблонов. Если ошибка не перехвачена ни на одном из уровней, обработка шаблона завершается, и метод Template Toolkit возвращает ложное значение.

```
[% TRY %]
 ...
 [% TRY %]
  [% INCLUDE $user.header %]
 [% CATCH file %]
  [% INCLUDE header %]
 [% END %]
 ...
[% CATCH DBI %]
 [% INCLUDE database/error.html %]
[% END %]
```

В этом примере внутренний блок TRY используется чтобы гарантировать правильную работу первой директивы INCLUDE. Переменная user.header используется для определения имени включаемого шаблона, и она может содержать имя несуществующего файла, или же возможно этот шаблон содержит неправильные директивы. Если INCLUDE выполняется неудачно с ошибкой 'file', то CATCH во внутреннем блоке ее перехватит и включит вместо ошибочного шаблона файл по умолчанию 'header'. Любая ошибка DBI внутри внешнего блока TRY будет перехвачена в соответствующем блоке CATCH и вызовет обработку шаблона 'database/error.html'. Необходимо учитывать, что включаемые шаблоны наследуют все определенные в данный момент переменные, следовательно, эти файлы ошибок могут получить информацию о только что перехваченном исключении через переменную 'error'. Например:

```
'database/error.html':

 <h2>Database Error</h2>
 A database error has occurred: [% error.info %]
```

Дополнительно можно определить блок FINAL, который всегда обрабатывается вне зависимости от результат обработки блоков TRY и/или CATCH. Если исключение не перехвачено, блок FINAL обрабатывается до перехода в окружающий блок или возврата в вызывающую процедуру.

```
[% TRY %]
 ...
[% CATCH this %]
 ...
[% CATCH that %]
 ...
[% FINAL %]
 All done!
[% END %]
```

Вывод блока TRY остается невредимым до точки, где произошло исключение. Например, этот шаблон:

```
[% TRY %]
 This gets printed
 [% THROW food 'carrots' %]
 This doesn't
[% CATCH food %]
 culinary delights: [% error.info %]
[% END %]
```

сгенерирует следующий вывод:

```
This gets printed
culinary delights: carrots
```

Директиву CLEAR можно использовать в блоках CATCH или FINAL для очистки всего вывода, созданного блоком TRY.

```
[% TRY %]
 This gets printed
 [% THROW food 'carrots' %]
 This doesn't
```

```
[% CATCH food %]
 [% CLEAR %]
 culinary delights: [% error.info %]
[% END %]
```

Вывод:

```
culinary delights: carrots
```

Возбудить исключение внутри шаблона можно с помощью директивы THROW. Первый параметр - тип исключения, который не нужно заключать в кавычки (но можно, как и в INCLUDE), следом идет соответствующее сообщение об ошибке, которое может быть любым допустимым выражением: строкой в кавычках, переменной и т.д.

```
[% THROW food "Missing ingredients: $recipe.error" %]
[% THROW user.login 'no user id: please login' %]
[% THROW $myerror.type "My Error: $myerror.info" %]
```

Кроме того, директиве THROW можно передать дополнительные или именованные параметры, если вы хотите передать через поле ошибки 'info' больше, чем простое текстовое сообщение.

```
[% THROW food 'eggs' 'flour' msg='Missing Ingredients' %]
```

В этом случае поле ошибки 'info' будет хешем, содержащим именованные параметры, в рассматриваемом случае 'msg' => 'Missing Ingredients', и элемент 'args', содержащий список дополнительных аргументов, в нашем случае 'eggs' и 'flour'. Поле ошибки 'type' остается без изменений, здесь оно устанавливается в 'food'.

```
[% CATCH food %]
 [% error.info.msg %]
 [% FOREACH item = error.info.args %]
  * [% item %]
 [% END %]
[% END %]
```

Полученный вывод:

```
Missing Ingredients
* eggs
* flour
```

7.2.3 Переменные

В разделе описываются различные переменные шаблона и доступ к ним через директивы Template Toolkit.

7.2.3.1 Переменные шаблона

Имена переменных могут содержать цифры, латинские буквы и знаки подчеркивания. Они могут быть записаны в нижнем, верхнем или смешанном регистрах, хотя общепринятой договоренностью в Template Toolkit является использование нижнего регистра. Тем не менее регистр важен, и 'foo', 'Foo' и 'FOO' являются именами разных переменных. Имена в верхнем регистре разрешены, но их использование не рекомендуется из-за возможных конфликтов с существующими или будущими зарезервированными словами. Такими являются:

```
GET CALL SET DEFAULT INSERT INCLUDE PROCESS WRAPPER

IF UNLESS ELSE ELSIF FOR FOREACH WHILE SWITCH CASE
USE PLUGIN FILTER MACRO PERL RAWPERL BLOCK META
TRY THROW CATCH FINAL NEXT LAST BREAK RETURN STOP
CLEAR TO STEP AND OR NOT MOD DIV END
```

Значениями переменных могут быть: обычные скаляры, ссылки на массивы и хеши.

7.2.3.2 Скаляры

Доступ к обычным скалярным переменным обеспечивается простым указанием их имен.

```
[% article %]
```

7.2.3.3 Ссылки на хеши

К элементам хеша можно обратиться указанием ссылки на хеш и ключ, отделенный оператором '.'.

```
[%
home = 'http://www.myserver.com/homepage.html';
page = {
 'this' => 'mypage.html',
 'next' => 'nextpage.html',
 'prev' => 'prevpage.html'
};
%]
```

шаблон:

```
<a href="[% home %]">Home</a>
<a href="[% page.prev %]">Previous Page</a>
<a href="[% page.next %]">Next Page</a>
```

вывод:

```
<a href="http://www.myserver.com/homepage.html">Home</a>
<a href="prevpage.html">Previous Page</a>
<a href="nextpage.html">Next Page</a>
```

Чтобы получить доступ к элементу хеша с ключом, значение которого сохранено в другой переменной, используйте '$' перед переменной, чтобы вычислить ее перед использованием (смотри Интерполяция переменных).

```
[% pagename = 'next' %]

[% page.$pagename %] # тоже что и [% page.next %]
```

Когда установлена переменная, которая содержит элементы пространств имен (т.е. имеет один и более символов '.' в имени), любые хеши, необходимые для представления промежуточных пространств имен, будут созданы автоматически. В следующем примере хеш 'product' будет автоматически создан, даже если он не был определен ранее.

```
[% product.id    = 'XYZ-2000'
product.desc  = 'Bogon Generator'
product.price = 666
%]

The [% product.id %] [% product.desc %] costs $[% product.price %].00
```

вывод:

```
The XYZ-2000 Bogon Generator costs $666.00
```

Можно использовать конструкцию '{' ... '}', чтобы явно определить хеш и присвоить его переменной. Учтите, что запятые не являются обязательными между парами ключ/значение и вместо '=>' можно использовать '='.

```
[% product = {
 id    => 'XYZ-2000',
 desc  => 'Bogon Generator',
 price => 666,
}
%]
```

7.2.3.4 Ссылки на массивы

Оператор точка также используется для доступа к элементам массива.

```
[%
people => [ 'Tom', 'Dick', 'Larry' ];
%]
```

шаблон:

```
[% people.0 %] # Tom

[% people.1 %] # Dick

[% people.2 %] # Larry
```

Для прохода по элементам массива используется директива FOREACH.

```
[% FOREACH person = people %]
 Hello [% person %]
[% END %]
```

вывод:

```
Hello Tom
Hello Dick
Hello Larry
```

Массивы можно создавать по месту, с использованием обычной конструкции анонимного массива '[' ... ']'. Запятые между элементами не являются обязательными.

```
[% cols = [ 'red', 'green', 'blue' ] %]
[% FOREACH c = cols %]
 ...
```

или:

```
[% FOREACH c = [ 'red', 'green', 'blue' ] %]
 ...
```

Простую последовательность чисел можно создать с использованием оператора '..':

```
[% n = [ 1 .. 4 ] %] # n принимает значения [ 1, 2, 3, 4 ]
[% x = 4
y = 8
z = [x..y]
%] # z принимает значения [ 4, 5, 6, 7, 8 ]
```

7.2.3.5 Интерполяция переменных

Template Toolkit использует '$' для однозначного указания того, что переменная должна быть вычислена на месте. Чаще всего это можно встретить в строках в двойных кавычках:

```
[% fullname = "$honorific $firstname $surname" %]
```

Те же правила действуют внутри директив. Если переменной предшествует '$', она перед использованием заменяется на свое значение. Наиболее часто это используется для получения элемента хеша, когда значение ключа сохранено в переменной.

```
[% uid = 'abw' %]

[% userlist.$uid %] # тоже что и 'userlist.abw'
```

Фигурные скобки используются при необходимости для выделения имен интерполируемых переменных.

```
[% userlist.${me.id}.name %]
```

Такие директивы как INCLUDE и PROCESS, которые принимают в качестве первого аргумента имя шаблона, для удобства автоматически заключают его в двойные кавычки.

```
[% INCLUDE foo/bar.txt %]
```

эквивалентно:

```
[% INCLUDE "foo/bar.txt" %]
```

Чтобы включить (INCLUDE) шаблон, чье имя сохранено в переменной, можно просто указать имя переменной с префиксом '$', и оно будет вычислено.

```
[% myfile = 'header' %]
[% INCLUDE $myfile %]
```

эквивалентно:

```
[% INCLUDE header %]
```

7.2.4 Виртуальные методы

Template Toolkit предоставляет виртуальные методы для работы со значениями переменных. Этот раздел описывает различные виртуальные методы, которые можно применять к скалярам, значениям массивов и хэшей.

7.2.4.1 Виртуальные методы для работы со скалярами

defined

Возвращает истинное значение если значение определенно.

```
[% user = get_user(uid) IF uid.defined %]
```

length

Возвращает длину строкового представления элемента:

```
[% IF password.length < 8 %]
 Password too short, dumbass!
[% END %]
```

repeat(n)

Повторяет строку указанное количество раз.

```
[% name = 'foo' %]
[% name.repeat(3) %] # foofoofoo
```

replace(search, replace)

Выводит строку с заменой всех вхождений первого аргумента (заданного как регулярное выражение) значением второго аргумента.

```
[% name = 'foo, bar & baz' %]
[% name.replace('\W+', '_') %] # foo_bar_baz
```

match(pattern)

Метод пытается сопоставить со строкой шаблон, передаваемый в качестве аргумента. Если шаблон встречается в строке, метод возвращает ссылку на массив всех строк, захваченных в скобках внутри шаблона.

```
[% name = 'Larry Wall' %]
[% matches = name.match('(\w+) (\w+)') %]
[% matches.1 %], [% matches.0 %] # Wall, Larry
```

Если шаблон не соответствует строке, метод возвращает ложное значение, что предпочтительнее, чем возвращение ссылки на пустой список, который Template Toolkit воспринимают как истинное значение. Это позволяет использовать подобные выражения:

```
[% "We're not worthy!" IF name.match('Larry Wall') %]
[% IF (matches = name.match('(\w+) (\w+)')) %]
 pattern matches: [% matches.join(', ') %]
[% ELSE %]
 pattern does not match
[% END %]
```

Любые модификаторы регулярных выражений, такие как '/s', следует добавлять в регулярные выражения с использованием синтаксиса '(?s)'. Например, для того, чтобы игнорировать «пробельные символы» в регулярном выражении (модификатор '/x') используется:

```
[% re = '(?x)(\w+)[ ](\w+)';
matches = name.match(re);
```

```
%]
```

split(pattern)

Вызывает функцию split() для разбиения строки на массив строк.

```
[% FOREACH dir = mypath.split(':') %]
 [% dir %]
[% END %]
```

7.2.4.2 Виртуальные методы для работы с хэшами

keys, values, each

Обычные операторы для работы с хэшами, возвращающие список ключей, список значений или список пар ключ-значение. Следует обратить внимание на использование префикса '$' перед переменной 'key' в следующем примере. Это необходимо для интерполяции переменной (т.е. замены значением) перед использованием.

```
[% FOREACH key = product.keys %]
 [% key %] => [% product.$key %]
[% END %]
```

sort, nsort

Возвращает список ключей, отсортированных по соответствующим значениям хэша по алфавиту (sort) или по числовым значениям (nsort).

```
[% FOREACH n = phones.sort %]
 [% phones.$n %] is [% n %],
[% END %]
```

defined, exists

Возвращает истинное или ложное значение в зависимости от того, определен или существует соответственно элемент хэша, обозначенный ключом, переданным в качестве аргумента методу.

```
[% hash.defined('somekey') ? 'yes' : 'no' %]
[% hash.exists('somekey') ? 'yes' : 'no' %]
```

size

Возвращает количество пар ключ-значение в хэше.

7.2.4.3 Виртуальные методы для работы с массивами

first, last

Возвращают первый/последний элементы массива. Элемент из массива не удаляется.

```
[% results.first %] to [% results.last %]
```

Если в качестве аргумента передается целое число, методы возвращают первые 'n' или последние 'n' элементов соответственно:

```
The first 5 results are [% results.first(5).join(", ") %].
```

size, max

Возвращают размер массива (количество элементов) и наибольшее значение индекса (size - 1), соответственно.

```
[% results.size %] search results matched your query
```

reverse

Возвращает элементы массива в обратном порядке.

```
[% FOREACH s = scores.reverse %]
 ...
[% END %]
```

join

Соединяет элементы массива в одну строку и использованием функции join.

```
[% items.join(', ') %]
```

grep

Возвращает список элементов массива соответствующих шаблону регулярного выражения.

```
[% FOREACH directory.files.grep('\.txt$') %]
 ...
[% END %]
```

sort, nsort

Возвращает элементы отсортированными по алфавиту (sort) или по числовому значению (nsort).

```
[% library = books.sort %]
```

Методу можно передать в качестве аргумента ключ поиска. Если элементы массива ссылки на хэши, ключ поиска будет использован для извлечения значения из хэша, которое затем будет использовано при сравнении.

```
[% library = books.sort('author') %]
```

unshift(item), push(item)

Добавляет элемент в начало/конец массива.

```
[% mylist.unshift('prev item') %]
[% mylist.push('next item')    %]
```

shift, pop

Удаляет первый/последний элементы массива и возвращает его.

```
[% first = mylist.shift %]
[% last  = mylist.pop   %]
```

unique

Возвращает список уникальных элементов массива в том же порядке, что и в оригинальном массиве.

```
[% mylist = [ 1, 2, 3, 2, 3, 4, 1, 4, 3, 4, 5 ] %]
[% numbers = mylist.unique %]
```

Поскольку существует возможность явно отсортировать массив, нет необходимости сортировать его до того как уникальные элементы будут извлечены.

```
[% numbers = mylist.unique.sort %]
```

slice(from, to)

Возвращает срез элементов массива между границами, установленными аргументами. Если второй аргумент 'to' опущен, то срез делается до последнего элемента массива. Оригинальный массив остается без изменений.

```
[% first_three = list.slice(0,2) %]
[% last_three  = list.slice(-3, -1) %]
```

splice(offset, length, list)

Позволяет выборочно удалять и/или заменять элементы в массиве. Он удалет 'length' элементов из массива, начиная с 'offset' и заменяет их элементами массива 'list'.

```
[% play_game = [ 'play', 'scrabble' ];
```

```
ping_pong = [ 'ping', 'pong' ];
redundant = play_game.splice(1, 1, ping_pong);
redundant.join; # scrabble
play_game.join; # play ping pong
%]
```

Метод возвращает список вырезанных элементов.

Кроме передачи ссылки на массив с элементами для замены, можно передавать элементы этого массива один за другим.

```
[% list.splice(-1, 0, 'foo', 'bar') %]
```

При передаче одного элемента в качестве значения для замены необходимо учитывать следующее: Еели это ссылка на массив, то она будет разрешена и будет использоваться содержимое всего массива, если нет, то она будет интерпретирована как одиночное значение. Для явного определения массива для замены необходимо использовать вокруг одиночного элемента квадратные скобки:

```
[% # вставить одиночный элемент an_item
list.splice(-1, 0, an_item);
# вставить элементы из another_list
list.splice(-1, 0, another_list);
# вставить ссылку на another_list
list.splice(-1, 0, [ another_list ]);
%]
```

7.2.5 Плагины

Следующие модули-плагины поставляются в составе Template Toolkit.

7.2.5.1 Date

ОБЗОР

```
[% USE date %]
# используем текущее время и формат по умолчанию
[% date.format %]
# время в секундах от точки отсчета или в виде строки 'h:m:s d-m-y'
[% date.format(960973980) %]
[% date.format('4:20:36 21/12/2000') %]
# определяем формат
[% date.format(mytime, '%H:%M:%S') %]
# определяем локаль
```

```
[% date.format(date.now, '%a %d %b %y', 'en_GB') %]
# именованные параметры
[% date.format(mytime, format = '%H:%M:%S') %]
[% date.format(locale = 'en_GB') %]
[% date.format(time   = date.now,
 format = '%H:%M:%S',
 locale = 'en_GB) %]

# определяем формат по умолчанию для плагина
[% USE date(format = '%H:%M:%S', locale = 'de_DE') %]
[% date.format %]
```

ОПИСАНИЕ

Плагин Date предоставляет простой способ форматировать строки со временем и датой с помощью функции POSIX strftime().

Плагин можно загрузить через знакомую директиву USE.

```
[% USE date %]
```

Это создает объект-плагин с именем по умолчанию 'date'. Другое имя можно определить следующим образом:

```
[% USE myname = date %]
```

Плагин предоставляет метод format(), который принимает значение времени, форматирующую строку и локаль. Все эти параметры не обязательны и имеют в качестве значений по умолчанию текущее системное время, формат '%H:%M:%S %d-%b-%Y' и текущую локаль соответственно. Значения по умолчанию для времени, формата и/или локали можно установить через именованные параметры в директиве USE.

```
[% USE date(format = '%a %d-%b-%Y', locale = 'fr_FR') %]
```

При вызове без параметров метод format() возвращает строку, представляющую текущее системное время, отформатированное с помощью strftime() в соответствие с форматом по умолчанию и локалью по умолчанию (которая может не совпадать с текущей, если установлена через директиву USE).

```
[% date.format %]
```

Плагин позволяет указывать время/дату как количество секунд от точки отсчета возвращения функцией time().

```
File last modified: [% date.format(filemod_time) %]
```

Также время/дату можно определить как строку вида 'h:m:s d/m/y'. В качестве разделителей можно использовать любые из символов : / - или пробел.

```
[% USE day = date(format => '%A', locale => 'en_GB') %]
[% day.format('4:20:00 9-13-2000') %]
```

Вывод:

```
Tuesday
```

Форматирующую строку также можно передавать методу format(), а следом можно указать локаль.

```
[% date.format(filemod, '%d-%b-%Y') %]
[% date.format(filemod, '%d-%b-%Y', 'en_GB') %]
```

Четвертый параметр позволяет выводить время по Гринвичу, в случае если на входе используется время в секундах с ключевого момента:

```
[% date.format(filemod, '%d-%b-%Y', 'en_GB', 1) %]
```

В случае если локальное время не совпадает с Гринвическим, указание в форматирующем параметре '%Z' (временная зона) приведет к неправильному результату.

Любые или все эти параметры могут быть именованными. Позиционные параметры должны всегда идти по порядку ($time, $format, $locale).

```
[% date.format(format => '%H:%M:%S') %]
[% date.format(time => filemod, format => '%H:%M:%S') %]
[% date.format(mytime, format => '%H:%M:%S') %]
[% date.format(mytime, format => '%H:%M:%S', locale => 'fr_FR') %]
[% date.format(mytime, format => '%H:%M:%S', gmt => 1) %]
...etc...
```

Метод now() возвращает текущее системное время в секундах с ключевого момента.

```
[% date.format(date.now, '%A') %]
```

7.2.5.2 Dumper

Это очень простой интерфейс-плагин Template Toolkit к модулю Data::Dumper. Объект Dumper создается следующей директивой:

```
[% USE Dumper %]
```

Как у всех стандартных плагинов можно указать его имя в нижнем регистре:

```
[% USE dumper %]
```

Объект Dumper поддерживает метод dump(), который выводит в шаблоне содержимое переданных методу объекта переменных.

dump()

Генерирует необработанный текстовый дамп структуры данных полученных переменных.

```
[% USE Dumper %]
[% Dumper.dump(myvar) %]
[% Dumper.dump(myvar, yourvar) %]
```

7.2.5.3 Format

```
[% USE format %]
[% commented = format('# %s') %]
[% commented('The cat sat on the mat') %]
[% USE bold = format('<b>%s</b>') %]
[% bold('Hello') %]
```

Плагин format создает функции, которые форматируют текст в соответствии с форматирующей строкой printf().

7.2.5.4 String

ОБЗОР

```
# создание объектов String с помощью директивы USE
[% USE String %]
[% USE String 'initial text' %]
[% USE String text => 'initial text' %]
# или из существующего объекта String через new()
[% newstring = String.new %]
[% newstring = String.new('newstring text') %]
[% newstring = String.new( text => 'newstring text' ) %]
# или из существующего объекта String через copy()
[% newstring = String.copy %]
# добавление текста к строке
[% String.append('text to append') %]
# форматирование с выравниванием влево, вправо или по центру
[% String.left(20) %]
[% String.right(20) %]
[% String.center(20) %]   # американская нотация
[% String.centre(20) %]   # европейская нотация
```

ОПИСАНИЕ

Этот модуль реализует класс String для выполнения строковых операций с текстом в объектно-ориентированном стиле.

Создать объект String можно с помощью директивы USE, добавив любой начальный текст в качестве аргумента или именованного параметра 'text'.

```
[% USE String %]
[% USE String 'initial text' %]
[% USE String text='initial text' %]
```

По умолчанию имя полученного объекта 'String', но объекту можно назначить другое имя переменной:

```
[% USE greeting = String 'Hello World' %]
```

Созданный объект String можно использовать в качестве прототипа для создания других объектов String посредством метода new().

```
[% USE String %]
[% greeting = String.new('Hello World') %]
```

Методу new() также можно передать начальный текст в качестве аргумента или именованного параметра 'text'.

```
[% greeting = String.new( text => 'Hello World' ) %]
```

Можно вызвать метод copy(), чтобы создать новый объект String как копию оригинала.

```
[% greet2 = greeting.copy %]
```

Объект String имеет метод text(), который возвращает содержимое строки.

```
[% greeting.text %]
```

Тем не менее, можно просто вывести строку по имени объекта, позволив перегруженному оператору стрингификации (stringification) вызвать метод text() автоматически.

```
[% greeting %]
```

Таким образом, можно рассматривать объекты String в значительной степени как обычный текст, интерполируя его в другие строки, например:

```
[% msg = "It printed '$greeting' and then dumped core\n" %]
```

Кроме того доступно множество других методов для манипуляций со строками.

```
[% msg.append("PS  Don't eat the yellow snow") %]
```

Следует учитывать, что все методы оперируют и изменяют содержимое самой строки. Если необходимо работать с копией, необходимо предварительно её создать:

```
[% msg.copy.append("PS  Don't eat the yellow snow") %]
```

Эти методы возвращают ссылку на сам объект String. Это позволяет строить цепочки из нескольких методов.

```
[% msg.copy.append('foo').right(72) %]
```

МЕТОДЫ

new()

Создает новую строку, используя начальное значение, передаваемое как позиционный аргумент или именованный параметр 'text'.

```
[% USE String %]
[% msg = String.new('Hello World') %]
[% msg = String.new( text => 'Hello World' ) %]
```

copy()

Создает новый объект String, который содержит копию оригинальной строки.

```
[% msg2 = msg.copy %]
```

text()

Возвращает внутреннее значение строки. Оператор стрингификации (stringification) перегружен на вызов этого метода. Таким образом, следующие вызовы эквивалентны:

```
[% msg.text %]
[% msg %]
```

length()

Возвращает длину строки.

```
[% USE String("foo") %]
[% String.length %] # => 3
```

search($pattern)

Осуществляет в строке поиск регулярного выражения, заданного в $pattern, и возвращает истинное или ложное значение в зависимости от результата поиска.

```
[% item = String.new('foo bar baz wiz waz woz') %]
[% item.search('wiz') ? 'WIZZY! :-)' : 'not wizzy :-(' %]
```

split($pattern, $limit)

Разбивает строку по разделителю $pattern и опциональному параметру $limit (максимальное количество частей, на которые требуется разбить строку).

```
[% FOREACH item.split %]
 ...
[% END %]
[% FOREACH item.split('baz|waz') %]
 ...
[% END %]
```

push($suffix, ...) / append($suffix, ...)

Добавляет все аргументы в конец строки. Метод append() предоставляется в качестве псевдонима к push().

```
[% msg.push('foo', 'bar') %]
[% msg.append('foo', 'bar') %]
```

pop($suffix)

Удаляет с конца строки суффикс, передаваемый в качестве аргумента.

```
[% USE String 'foo bar' %]
[% String.pop(' bar')   %] # => 'foo'
```

unshift($prefix, ...) / prepend($prefix, ...)

Добавляет все аргументы в начало строки. Метод prepend() предоставляется в качестве псевдонима к unshift().

```
[% msg.unshift('foo ', 'bar ') %]
[% msg.prepend('foo ', 'bar ') %]
```

shift($prefix)

Удаляет с начала строки префикс, передаваемый в качестве аргумента.

```
[% USE String 'foo bar' %]
[% String.shift('foo ') %] # => 'bar'
```

left($pad)

Если длина строки меньше чем $pad, строка выравнивается влево и дополняется пробелами до длины $pad.

```
[% msg.left(20) %]
```

right($pad)

Аналогично left(), но строка выравнивается вправо и дополняется слева пробелами до длины $pad.

```
[% msg.right(20) %]
```

center($pad) / centre($pad)

Аналогично left() и right(), но строка выравнивается по центру и дополняется пробелами слева и справа до длины $pad. Метод centre() предоставляется в качестве псевдонима к center().

```
[% msg.center(20) %] # американская нотация
[% msg.centre(20) %] # европейская нотация
```

format($format)

Применяет к строке формат в стиле sprintf().

```
[% USE String("world") %]
[% String.format("Hello %s\n") %] # => "Hello World\n"
```

upper()

Преобразует строку в верхний регистр.

```
[% USE String("foo") %]
[% String.upper %] # => 'FOO'
```

lower()

Преобразует строку в нижний регистр.

```
[% USE String("FOO") %]
[% String.lower %] # => 'foo'
```

capital()

Преобразует в верхний регистр первый символ строки.

```
[% USE String("foo") %]
[% String.capital %] # => 'Foo'
```

Остальная часть строки остаётся без изменений. Гарантировано преобразовать всю строку в нижний регистр, а первую букву - в верхний можно следующим образом:

```
[% USE String("FOO") %]
[% String.lower.capital %] # => 'Foo'
```

chop()

Удаляет последний символ в строке.

```
[% USE String("foop") %]
[% String.chop %] # => 'foo'
```

chomp()

Удаляет завершающий символ новой строки.

```
[% USE String("foo\n") %]
[% String.chomp %] # => 'foo'
```

trim()

Удаляет начальные и завершающие пробельные символы в строке.

```
[% USE String("   foo   \n\n ") %]
[% String.trim %] # => 'foo'
```

collapse()

Удаляет начальные и завершающие пробельные символы в строке, а также сжимает в строке последовательности из нескольких пробельных символов в один пробел.

```
[% USE String(" \n\r  \t  foo   \n \n bar  \n") %]
[% String.collapse %] # => "foo bar"
```

truncate($length, $suffix)

Обрезает строку до длины $length символов.

```
[% USE String('long string') %]
[% String.truncate(4) %] # => 'long'
```

Если указан $suffix, он будет добавлен к обрезанной строке. В этом случае строка будет дополнительно укорочена на длину суффикса, чтобы модифицированная строка с суффиксом имела точно длину $length символов.

```
[% USE msg = String('Hello World') %]
[% msg.truncate(8, '...') %] # => 'Hello...'
```

replace($search, $replace)

Заменяет все вхождения $search в строке на $replace.

```
[% USE String('foo bar foo baz') %]
[% String.replace('foo', 'wiz')  %] # => 'wiz bar wiz baz'
```

remove($search)

Удаляет все вхождения $search в строке.

```
[% USE String('foo bar foo baz') %]
[% String.remove('foo ')  %] # => 'bar baz'
```

repeat($count)

Повторяет строку $count раз.

```
[% USE String('foo ') %]
[% String.repeat(3)  %] # => 'foo foo foo '
```

7.2.6 Директивы препроцессора

Все директивы препроцессора должны находится вне кода шаблонизатора, т.е вне последовательности символов '[%' и '%]'.

LOOKUP

Синтаксис:

```
#LOOKUP <Что искать> [<где искать> ...]
```

Ищет указанный сценарий начиная с указанного пути, поднимаясь по дереву вверх до корневого раздела, пока не найдет первый подходящий (сравните с LOCATE, см. далее). Вставляет данный сценарий в место директивы как новый [% BLOCK %] шаблонизатора, и производит его вызов через директиву шаблонизатора [% PROCESS %]. Если опустить параметр <где искать>, то поиск будет производится начиная с текущего каталога (.), то есть каталога, в котором расположен шаблон, из которого произведен вызов #LOOKUP.

Примеры:

```
#LOOKUP PPPOE/activate /BASE/ALCATEL_STD/ALCATEL_7302 /ADDONS/TEMP
```

Произведет поочередно поиск сценария activate по следующим путям:

/BASE/ALCATEL_STD/ALCATEL_7302/PPPOE/activate

/BASE/ALCATEL_STD/PPPOE/activate

/BASE/PPPOE/activate

/PPPOE/activate

/ADDONS/TEMP/PPPOE/activate

/ADDONS/PPPOE/activate

```
#LOOKUP functions.tpl
```

Произведет поочередно поиск сценария functions:

./functions.tpl

../functions.tpl

../../functions.tpl

…

/functions.tpl

LOCATE

Синтаксис:

```
#LOCATE <Что искать> [<где искать> ...]
```

Ищет указанный сценарий только в указанном разделе (разделах), не поднимаясь по дереву и не спускаясь по нему (сравни с LOOKUP, см. предыдущий). Вставляет данный сценарий в код как новый [% BLOCK %] шаблонизатора и производит его вызов через директиву шаблонизатора [% PROCESS %]. Если опустить параметр <где искать>, то поиск будет производится начиная с текущего каталога (.), то есть каталога, в котором расположен шаблон, из которого произведен вызов #LOCATE.

Например:

```
#LOCATE  functions.tpl /FUNCTIONS.dir
```

8 Постпроцессор

Директивы постпроцессора выполняет модуль ШПД на этапе отправки MML-инструкций или строк-запросов на сетевой элемент. Все директивы постпроцессора, как и MML-инструкции и строки-запросов, должны находится вне кода шаблонизатора, т.е вне последовательности символов '[%' и '%]'.

8.1 Комментарии

Синтаксис:

```
#some text
```

Комментарии должны начинаться с символа '#'. Данные строки будут проигнорированы (не будут отправлены на сетевой элемент) постпроцессором модуля ШПД, за исключением строк, содержащих директивы препроцессора, такие как '#LOOKUP' и '#LOCATE'.

8.2 Общие директивы постпроцессора

Директивы постпроцессора, представленные в данном разделе Руководства предназначены для использования в любых сценариях, в отличие от директив постпроцессора в сценариях проверки, которые описаны в разделе 8.3 (стр. 72).

FAIL

Синтаксис:

```
FAIL "some text"
```

Сообщает модулю ШПД завершить выполнение сценария с ошибкой и использовать текст «some text» как отображаемую причину ошибки. Если такая директива встречается в любом месте сценария, сценарий прекращает выполняться.

Пример:

```
FAIL "unknow board type"
```

SUCCESS

Синтаксис:

```
SUCCESS
```

Сообщает модулю ШПД успешно завершить выполнение сценария. Если такая директива встречается в любом месте сценария, сценарий прекращает выполняться.

GOTO

Синтаксис:

```
GOTO STEP<number>
```

Сообщает постпроцессору «перейти» к директиве STEP с номером <number>. Когда модуль ШПД встречает директиву GOTO, он запрашивает часть сценария (см. директиву STEP) передавая шаблонизатору контекст последней перед директивой GOTO выполненной MML-инструкции или строки-запроса. После выполнения необходимой части сценария выполнение продолжается со строки, следующей за директивой GOTO, а ответные сообщения сетевого элемента на MML-инструкции или строки-запроса, выполненные в вызываемой части сценария, добавляются к ответу общего сценария.

Директива STEP, на которую ссылается директива GOTO должна находиться в том же сценарии, что и директива GOTO!

Пример:

```
GOTO STEP2
```

STEP

Синтаксис:

```
STEP<number>
<some text>
END STEP<number>
```

Директива STEP является блоком сценария <some text>, который вызывается директивой GOTO. При использовании в данном блоке кода шаблонизатора в шаблонизатор передается контекст последней выполненной MML-инструкции или строки-запроса (переменная PMS__OUT). Внутри блока сценария не должно быть других блоков шаблонизатора, возможно использовать только переходы к другим блокам сценария с помощью директивы GOTO.

Пример:

```
info configure xdsl line 1/1/[%SLOT%]/[%PORT%] detail xml
GOTO STEP2


STEP2
 [%IF PMS__OUT.match('actual-type\s+:\s+nslt-a')%]
  GOTO STEP4
 [%ELSIF (PMS__OUT.match('actual-type\s+:\s+nalt-[cd]'))%]
  GOTO STEP3
 [%ELSE%]
  FAIL 'unknown board type'
```

```
 [%END%]
END STEP2
```

8.3 Специфические директивы постпроцессора в сценариях проверок

Данные директивы не являются обязательными в сценариях, и если сценарий их не содержит, то:

1. Если сценарий будет выполнен без ошибок, то в результатах проверка с именем сценария будет успешна.
2. Если сценарий будет выполнен с ошибками, то в результатах проверка с именем сценария будет не успешна.

Пример использования приведён в Приложение 3. Пример использования директив постпроцессора в сценарии проверок.

RFS_OK

Синтаксис:

```
RFS_OK
```

В описаниях результатов выполнения создаётся запись с именем сценария, в эту запись помещается результат проверки «успешно». Выполнение сценария прекращается.

RFS_STAGE_OK

Синтаксис:

```
RFS_STAGE_OK '<some stage name>'
```

В описаниях результатов выполнения создаётся запись с именем <some stage name>, в эту запись помещается результат проверки «успешно». Выполнение сценария не прекращается.

Используется для реализации нескольких проверок в одном сценарии.

RFS_FAIL

Синтаксис:

```
RFS_FAIL ['<some fail message>']
```

В описаниях результатов выполнения создаётся запись с именем сценария, в эту запись помещаются:

- Результат проверки «не успешно».

- В описании результата помещается сообщение <some fail message>, или SCENARIO FAILED при его отсутствии (по умолчанию).

Выполнение сценария прекращается.

RFS_STAGE_FAIL

Синтаксис:

```
RFS_STAGE_FAIL '<some stage name>' ['<some fail message>']
```

В описаниях результатов выполнения создаётся запись с именем <some stage name>, в эту запись помещаются:

- Результат проверки «не успешно».
- В описании результата помещается сообщение <some fail message>, или SCENARIO FAILED при его отсутствии (по умолчанию).

Выполнение сценария не прекращается.

Используется для реализации нескольких проверок в одном сценарии.

8.4 MML-инструкции и строки-запросов

Если строка не будет содержать какие-либо директивы постпроцессора, то постпроцессор посчитает, что это MML-инструкция или строка-запроса для сетевого элемента и выполнит ее. Каждая строка, содержащая MML-инструкцию или строку-запроса, после выполнения создает контекст, являющийся ответом сетевого элемента. Данный контекст можно использовать в директиве 'GOTO'.

9 Разбор ответа

После выполнения сценария модулем ШПД, используя код шаблонизатора, директивы препроцессора и общий контекст сценария (полный ответ сетевого элемента после выполнения всего сценария: переменная PMS__OUT), данный блок позволяет сформировать набор переменных для использования в блоке визуализации (см. п. 10).

Пример:

```
[%
CHECK_OPTIONS = {
 'admstate' => 1,
 'oprstate' => 1,
 'vlan' => 1,
 'pvc' => 1,
};
IF (PMS__OUT.match('admstate: down'));
 CHECK_OPTIONS.admstate = 0;
END;
%]
```

Далее, в блоке визуализации, данные переменные можно использовать так:

```
<span class="bgcolor" align="center">
[%IF (CHECK_OPTIONS.admstate)%]
 подключен
[%ELSE%]
 <font color='red'>ОТКЛЮЧЕН</font>
[%END%]
</span>
```

10 Визуализация (HTML)

Данный блок позволяет с помощью HTML и используя код шаблонизатора, директивы препроцессора и сформированные в блоке разбора ответа переменные создавать визуальное представление результата выполнения сценария (задаёт форму представления результата выполнения Команд экранных формах для работы оператора технической поддержки, см. Руководство оператора технической поддержки.

Пример:

```
<table border="0" cellpadding="5" cellspacing="0" width="100%">
 <tr>
  <td class="bgcolor" align="left">Статус порта</td>
  <td class="bgcolor" align="center">
   [%IF (CHECK_OPTIONS.admstate)%]
    подключен
   [%ELSE%]
    <font color='red'>ОТКЛЮЧЕН</font>
   [%END%]
  </td>
 </tr>
 <tr>
  <td class="bgcolor" align="left">Статус соединения</td>
   <td class="bgcolor" align="center">
   [%IF (CHECK_OPTIONS.oprstate)%]
    активен
  [%ELSE%]
    <font color='red'>>не активен</font>
   [%END%]
  </td>
 </tr>
</table>
```

11 XML

Применяется для формирования ответа внешней системе. Данный блок позволяет, используя код шаблонизатора, директивы препроцессора и сформированные в блоке разбора ответа переменные, формировать объект данных, который будет преобразован в XML, который будет включен в стандартный XML-формат результатов выполнения сценария в модуле ШПД. Объект должен создаваться в переменной INNER_XML

Пример:

```
[%
 INNER_XML = {
  'AdmState' => CHECK_OPTIONS.admstate,
  'OprState' => CHECK_OPTIONS.oprstate
 };
%]
```

Приложения

Приложение 1. Пример результатов выполнения проверок

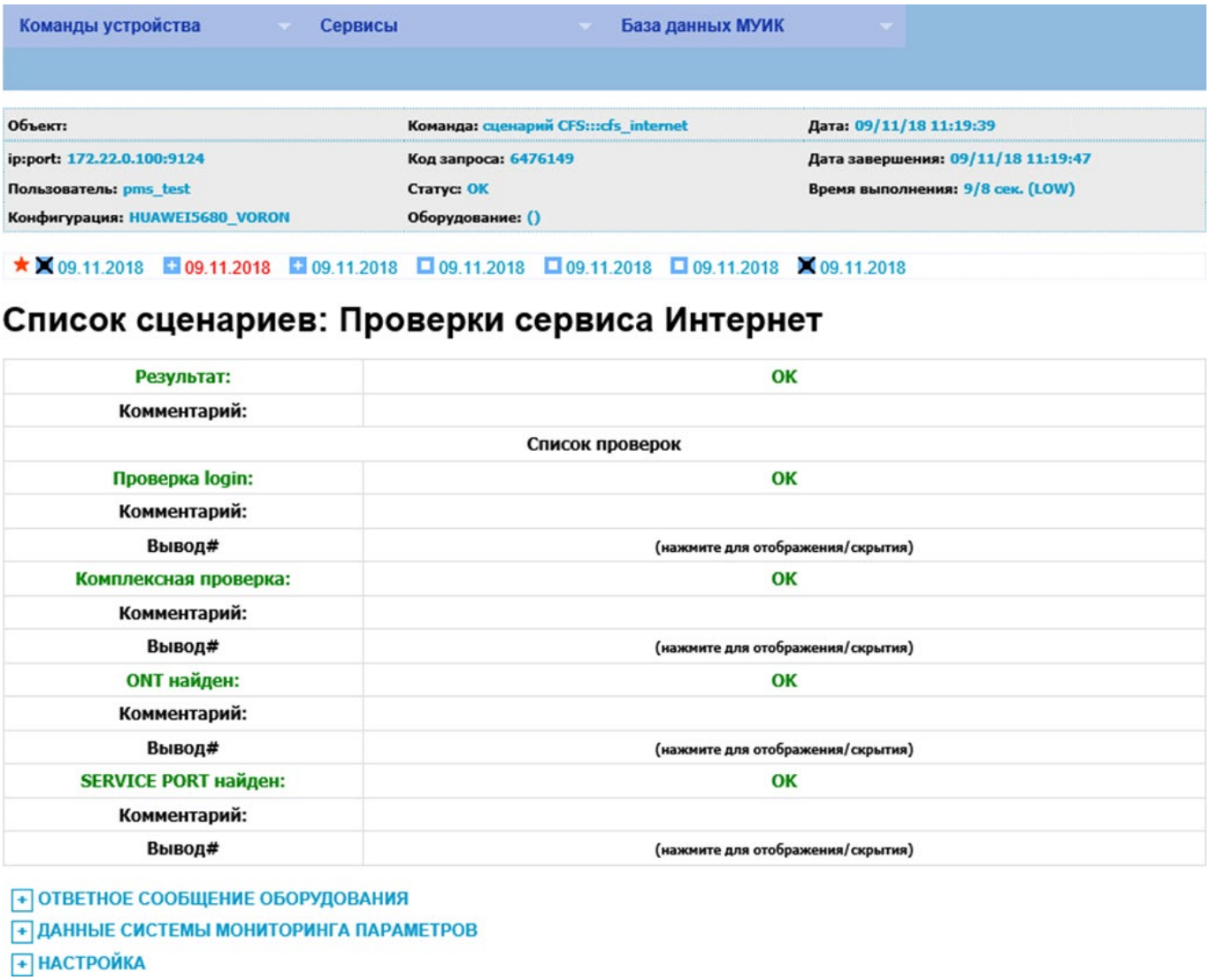

Рисунок 49 – Пример вывода на экран результатов успешной проверки

Приложение 2. Пример ответного сообщения XML с результатами проверок сервиса

```
<get_cfs_resultReturn xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <title>Проверки сервиса Интернет</title>
      <result>FAIL</result>
      <comment/>
      <rfs_list>
```

```
<RFSRes>
   <qid>6476147</qid>
   <title>Проверка login</title>
   <result>OK</result>
   <comment/>
</RFSRes>
<RFSRes>
   <qid>6476147</qid>
   <title>Комплексная проверка</title>
   <result>FAIL</result>
   <comment>Command failed</comment>
</RFSRes>
<RFSRes>
   <qid>6476149</qid>
   <title>Проверка login</title>
   <result>OK</result>
   <comment/>
</RFSRes>
<RFSRes>
   <qid>6476149</qid>
   <title>Комплексная проверка</title>
   <result>OK</result>
   <comment/>
</RFSRes>
<RFSRes>
   <qid>6476149</qid>
   <title>ONT найден</title>
   <result>OK</result>
   <comment/>
</RFSRes>
<RFSRes>
   <qid>6476149</qid>
   <title>SERVICE PORT найден</title>
```

```
          <result>OK</result>
          <comment/>
        </RFSRes>
    </rfs_list>
</get_cfs_resultReturn>
```

Приложение 3. Пример использования директив постпроцессора в сценарии проверок

```
interface gpon [% FRAME_ID %]/[%SLOT%] port [%PORT%] ont-auto-find enable
display ont info [%PORT%] all
GOTO STEP1;

quit

display service-port port [%FRAME_ID%]/[%SLOT%]/[%PORT%] ont [%ONT%] GOTO STEP2;
RFS_OK

STEP1; [%
ONTS = PMS_OUT.split("\n"); need_add = 1;
FOREACH CONT IN ONTS;
     IF (matches = CONT.match('0\/\s*[0-9]+\s+([0-9A-Z]+)\s+active')); IF (ONT_SN
          == matches.0);
               need_add = 0; LAST;
          END;
     END;
END;
IF (need_add)%]
RFS_FAIL 'ONT не найден' [%END%]
RFS_STAGE_OK 'ONT найден' END STEP1;

STEP2; [%
SRVS = PMS_OUT.split("\n"); FOREACH srv IN SRVS;
     IF (matches = srv.match('([0-9]+)\s+([0-9'));
        IF ( (SLOT == matches.2.replace('\s','')) && matches.1 == SVLAN); srv_id
            = matches.0;%]
RFS_STAGE_OK 'SERVICE PORT найден' [%LAST;
          END;
     END;
END%]
END STEP2;
```