Eclipse/ExDT – инструментарий для настройки интерактивных сред разработки (IDE)

Содержание

Введение

Данный документ описывает общие принципы и средства настройки интерактивных сред разработки (Interactive Development Environment – IDE), разрабатываемые компанией Excelsior по заказу Elphel, Inc. для поддержки разработки аппаратных средств, прежде всего на базе технологий FPGA и языка Verilog.

Требования

Одной из причин для начала работ в данном направлении стала потребность Elphel в удобном инструментарии разработчика аппаратных средств. Опыт использования других инструментальных средств выявил отсутствие универсальных технологий и безупречных инструментов в этой сфере производства, а также их недостаточную настраиваемость и комбинируемость друг с другом. Существующие программные разработки чаще всего представляют собой готовые пакеты технологических решений, предоставляемые компаниями-производителями hardware, которые ориентированы на работу именно с их средствами, только в рамках их технологий, и не допускают вообще или допускают ценой больших усилий интеграцию с другими инструментами. Альтернативные же инструменты, во-первых, часто покрывают лишь отдельные этапы технологии (например, Verilog-симуляцию), во-вторых, либо управляются только в командном режиме путем написания сложных скриптов, либо имеют специфическую обособленную интерактивную оболочку. В любом случае, если разработчик использует собственную уникальную технологию, отдельные части которой поддерживаются разными инструментами, он не имеет возможности прогона всех технологических этапов в рамках единой среды разработки и зачастую тратит усилия на автоматизацию своих производственных процессов путем создания скриптов на командном языке своей ОС.

Поэтому к создаваемой среде разработчика были предъявлены следующие требования:

- *расширяемость*: возможность интеграции как можно более широкого спектра существующих и новых программных средств;
- *поливариантность*: возможность настройки нескольких одновременно доступных вариантов технологических процессов, в том числе с использованием альтернативных программных средств;
- *гибкость*: возможность изменения настройки в рамках конкретной инсталляции и для конкретного пользователя;
- *простота*: возможность выполнения перечисленных работ самим конечным пользователем-разработчиком;
- *переносимость*: возможность переноса разработки на различные операционные платформы, в частности, Windows и Linux.

Все эти свойства в дальнейшем рассматриваются как аспекты ***пользовательской настройки*** IDE.

В качестве базы для реализации была выбрана расширяемая IDE Eclipse.

Eclipse

Eclipse является открытой оболочкой, которую можно настроить на широкий спектр технологий разработки программного обеспечения. Уже существует ряд специализированных систем программирования на базе Eclipse для многих известных языковых технологий (C++, Java и др.) и инструментальных пакетов.

Настройка Eclipse – программируемая: она выполняется путем создания на языке Java программных расширений – ***плагинов (plug-ins)***, добавляющих в Eclipse новые возможности путем реализации программных интерфейсов различных функциональных элементов оболочки. Через интерфейсы же плагины могут пользоваться другими функциями оболочки.

Возможности программируемой настройки весьма широки, однако, разработка плагина для Eclipse, даже несмотря на открытость оболочки, остается трудоемкой программистской работой, непосильной конечному пользователю-разработчику. То же касается и доработки готовых плагинов, например, при необходимости добавить новые инструменты в среду или изменить существующие.

Eclipse/VDT

Целевой вариант пользовательской инструментальной среды, ориентированный на технологии языка Verilog и удовлетворяющий выдвинутым требованиям настраиваемости, разрабатывается как плагин **Verilog Development Tools** для Eclipse (**Eclipse/VDT**).

Документ *«Пользовательское описание Eclipse/VDT»* содержит описание этого плагина. Полезно предварительно ознакомиться с ним для получения более наглядного представления о возможностях плагина.

Плагин Eclipse/VDT позволяет интегрировать в Eclipse третьесторонние прикладные программные инструменты (утилиты и пакеты утилит) для разработки пользовательского проекта. Действия по интеграции и управлению режимами работы этих инструментов, называются ***настройкой инструментов***.

В Eclipse/VDT предусмотрены два уровня настройки инструментов:

- ***оперативная*** настройка режима работы инструмента – выполняется с помощью элементов управления, предоставляемых IDE для данного проекта: в диалогах настройки свойств можно изменять значения настроечных параметров, которые затем передаются утилитам через командные строки и управляющие файлы; этот вид настройки традиционен для диалоговых приложений;
- ***функциональная*** настройка состава инструментов в IDE и их интерфейсов управления – выполняется с помощью редактируемого XML-описания, в котором можно задать состав и свойства конфигурируемых управляющих параметров пользовательских утилит, вид диалогов настройки, форматы командных строк и файлов, составить меню инструментов; этот вид настройки является отличительной чертой Eclipse/VDT.

Функциональная настройка выполняется ***динамически***: она не требует перепрограммирования, перетрансляции и переподключения плагина – изменения в XML-файлах вступают в силу при следующем вызове Eclipse.

В процессе разработки этого плагина стало ясно, что реализуемые в нем возможности функциональной настройки выходят далеко за рамки целевой области применения: они могут эффективно применяться для поддержки различных технологий разработки программных средств широкого спектра назначений.

Поэтому средства функциональной настройки оказалось целесообразным вынести в отдельный плагин инструментального назначения – Eclipse/ExDT, в рамках которого они и будут описаны ниже.

Под Eclipse/VDT в данный момент понимается версия Eclipse/ExDT, программно специализированная для технологий разработки на языке Verilog. В дальнейшем предполагается разделить эти два плагина так, чтобы Eclipse/ExDT стал общим инструментарием для создания настраиваемых IDE, а Eclipse/VDT – одной из возможных программно-функциональных специализаций его для конкретной прикладной области.

Далее все вопросы функциональной настройки рассматриваются как свойства Eclipse/ExDT.

Eclipse/ExDT

Плагин **Extensible Development Tools** для Eclipse (**Eclipse/ExDT**) разрабатывается как многопрофильный инструментарий для создания настраиваемых IDE на базе Eclipse.

Возможности функциональной настройки в Eclipse/ExDT, реализуемые динамически интерпретируемым XML-описанием настроек, занимают промежуточный уровень между

широкими возможностями программируемой настройки в Eclipse и узкими возможностями оперативной настройки в Eclipse/VDT и других специализированных IDE.

В отличие от специализированных IDE, Eclipse/ExDT не предоставляет пользователю готовых программных инструментов для решения его прикладных задач. Он содержит только интерфейсную оболочку и инструментальное ядро, к которым пользователь может сам добавить требуемое функциональное наполнение – избранные прикладные программы и пакеты, создав тем самым специализированную IDE (впрочем, легко переспециализируемую).

Единственным требованием, определяющим возможность интеграции той или иной программы в ExDT, является наличие у неё командного интерфейса управления.

TSL

Функциональная настройка Eclipse/ExDT, т.е. описание доступных пользователю программных средств и способов управления ими, выполняется с помощью внутреннего ***языка настройки инструментов TSL (Tools Specification Language)***, основанном на XML. TSL-спецификация полностью открыта для пользователя и не требует предварительной компиляции плагина: она интерпретируется при загрузке Eclipse/ExDT.

TSL является метаязыком, описывающим, в действительности, три взаимосвязанных интерфейса управления:

- ***интерфейс пользовательского управления (входной)***, связанный с визуализацией и вводом параметров настройки в диалогах;
- ***интерфейс командного управления программой (выходной)***, связанный с описанием командного языка управления программой и передачей ей параметров при вызове;
- ***интерфейс представления данных (внутренний)***, связанный с внутренней организацией элементов TSL-спецификации и связей между ними.

В дальнейшем понятия и пояснения, относящиеся к каждому из этих интерфейсов, выделяются в тексте соответствующим цветом.

Основные компоненты технологии Eclipse/ExDT

Здесь вводятся термины и определения для основных понятий, используемых в Eclipse/ExDT, а также перечисляются основные компоненты плагина с пояснением их назначения.

Проекты и технологии

Вся работа пользователя в Eclipse осуществляется в рамках ***рабочего проекта (work project)*** и связана с обработкой включенных в него данных с помощью некоторого набора инструментов. Состав и свойства этих инструментов зависят от типа проекта, который определяется плагином.

Изготовление проекта (project build/design) заключается в получении требуемого набора выходных данных из начального набора входных данных. Изготовление может выполняться пошагово, с образованием промежуточных данных. Изготовление может вестись вручную, путем непосредственного редактирования данных, автоматически, с помощью программных инструментов, а также смешанным образом.

(Замечание о терминах: в терминологии конструирования микросхем проект чаще называется *design*, а его изготовление – *implement design*;поэтому в VDT используется этот термин).

Технологический процесс (build/design flow) – это цепочка элементарных шагов обработки данных, соответствующая одному из способов изготовления проекта.

Технология разработки (build/design technique) проекта – это вся совокупность приемов и инструментов, применяющихся при изготовлении проекта.

Перспектива и инструменты

Перспектива (perspective) определяет доступный пользователю набор инструментов и управляющих средств, вид панелей управления и их расположение на экране Eclipse.

В Eclipse/ExDT есть три вида инструментов:

- ***навигаторы (navigators)***, которые визуализируют содержимое проекта в виде различных ***представлений (views)*** и позволяют выбирать текущую компоненту для обработки;
- языково-ориентированные ***редакторы*** текстовых файлов ***(editors)***;
- ***прикладные инструменты (application tools)***, вызывающие внешние утилиты для автоматической обработки данных.

Данные, навигаторы и редакторы

Основными объектами данных в Eclipse/ExDT являются ***файлы*** различных типов. Тип файла соответствует языку или формату содержимого файла и определяется по расширению его имени. Представление проекта в виде списка файлов реализуется файловым навигатором **File View**.

Другими объектами данных могут быть различные языковые компоненты программ. Для их представления могут программироваться специальные навигаторы. Например, для Eclipse/VDT разработан навигатор **Module View**, реализующий представление проекта в виде иерархии Verilog-модулей.

Редакторы файлов ассоциированы с файлами определенных типов и могут иметь языково-ориентированные черты, такие как цветовая раскраска лексем (syntax hilighting), контекстные подсказки, поддержка навигации по списку ошибок и др.

В данный момент навигаторы и редакторы для Eclipse/ExDT и его специализацированных версий выбираются из числа существующих в мире или программируются отдельно. Их интеграция выполняется программным путем.

Меню прикладных инструментов

В Eclipse/ExDT имеется иерархическое ***меню инструментов (design menu)***, в котором отдельными кнопками представлены ***инструменты (tools)***, представляющие собой вызовы внешних утилит с сохраняемыми параметрами.

Термин «инструмент» соотносится не с самой программой, а с некоторым видом работ, выполняемых ею в одном из возможных режимов: инструмент – это элементарный, логически обособленный шаг в технологическом процессе, который может реализовываться как отдельной программой, так и последовательностью программ, объединенных в batch-файл.

Меню инструментов является настраиваемым элементом текущего проекта. Оно может строиться индивидуально для каждой технологии разработки. Предполагается, что с пакетами инструментов поступают типовые меню, описывающие технологии от производителя, а на их базе программист строит свои версии меню, учитывающие особенности технологии в его проекте.

Описания структуры и составы меню выполняются на TSL и интегрируются в среду разработки динамически.

Доступность (availability) каждого инструмента в меню в данный момент времени зависит от того, разрешено ли его применение к текущему объекту данных.

Для каждого инструмента в меню можно выполнить следующие действия:

- ***оперативную настройку*** – действия по изменению значений управляющих параметров;
- ***вызов (run)*** – формирование командной строки и командных файлов для программы, ассоциированной с инструментом, запуск этой программы, получение протокола ее вывода в отдельном окне консоли.

Дополнительно в меню можно выполнить следующие действия:

- ***инсталляционную настройку*** свойств Eclipse/ExDT, пакетов прикладных программ и текущего проекта;
- оперативную настройку свойств прикладных программ, единых для пакета или проекта.

Параметры и диалоги настройки инструментов

Интерфейс пользовательского управления определяет способы оперативной настройки инструментов пользователем в процессе работы.

Каждый инструмент управляется собственным набором ***параметров (parameters)***, для которых можно задавать одно из возможных ***значений (values)*** в соответствии с их ***типом (type)***. В ***диалогах настройки параметров (setup dialogs)*** можно изменять текущие значения доступных параметров.

В диалогах настройки параметры могут быть разделены по смысловым ***группам (groups)***, каждой из которых соответствует отдельная ***закладка (tab)*** окна диалога. Каждый параметр в диалоге представлен поясняющей ***пометкой (label)*** и ***полем ввода*** значения ***(input control)***. Вид поля ввода зависит от типа параметра и может быть одним из следующих:

- ***флажок (check-box)*** – для логических параметров;
- ***редактируемая строка (editable text box)*** – для текстовых и числовых параметров;
- ***выпадающий список выбора (drop-down selection list box)*** – для перечислимых параметров (с фиксированным набором значений);
- ***кнопка обзора (browse button)*** – для выбора файлов и директорий;
- ***диалог ввода списочного значения (list value input dialog)*** – для параметров, допускающих список элементов в качестве значения.

Существует несколько уровней настройки параметров (см. раздел *«Контексты...»*), каждому из которых соответствует отдельный диалог. Все диалоги настройки доступны из меню инструментов.

Утилиты и языки командного управления

Утилита (utility) – это отдельная программа, реализующая работу одного или нескольких инструментов. Утилиты могут быть ***обособленными (stand-alone)*** или объединяться в ***пакеты (packages)***, поддерживающие некоторую сквозную технологию разработки. Утилиты в пакете могут иметь общее ***местоположение (location)*** (что облегчает их настройку) и единообразие языков управления (что позволяет упростить их описание).

В качестве утилиты к Eclipse/ExDT может быть подключена любая программа, вызов которой может быть описан средствами интерфейса командного управления.

(Это не обязательно должно быть консольное приложение: программа может выдавать свои графические диалоги, однако какое-либо их сопряжение с диалогами Eclipse не обеспечивается.)

Интерфейс командного управления определяет способы вызова утилиты и передачи ей управляющих параметров.

При вызове утилиты ей передается управляющая информация в виде одной или нескольких ***управляющих строк (control lines)***, носителями которых являются единственная ***командная строка (command line)*** и, возможно, один или несколько ***командных файлов (command files)***. Содержание и вид командных строк определяются языком управления утилиты и спецификацией вызова.

Управляющая строка утилиты составляется из набора ***опций (options)*** с заданными ***установками (settings)***. Каждая разрешенная комбинация разрешенных установок опций задает один из возможных ***режимов работы (operation mode)*** утилиты.

Определенный для консольного приложения ***стандартный поток вывода (standard output stream, stdout)*** утилиты перенаправляется в отдельное ***консольное окно (console window)*** Eclipse для просмотра и возможной обработки.

Соответствие входных параметров и выходных опций

При вызове инструмента происходит запуск утилиты, реализующей его функцию; при этом входные параметры преобразуются в опции утилиты, а значения параметров – в установки опций. Способ преобразования – ***выходной формат (output format)*** – выбирается для каждого параметра индивидуально из числа предусмотренных общих схем.

Таким образом, между параметрами инструментов и опциями утилит есть соответствие, которое чаще всего взаимнооднозначно. Это соответствие устанавливается средствами TSL с помощью объекта `parameter`, которое связывает воедино ***входное представление*** параметра (способ его показа и ввода в диалоге настройки), ***выходное представление*** (способ преобразования в установку опции) и ***внутреннее представление*** (атрибуты и связи с другими параметрами и объектами TSL-спецификации).

Однако, не все предусмотренные опции утилиты должны в обязательном порядке настраиваться через параметры: некоторые из них могут передаваться с константными установками, фиксирующими тот режим работы утилиты, который реализует возможности данного инструмента. Иногда отсутствие опции в управляющей строке может означать ее передачу со значением по умолчанию.

Контексты и правила видимости параметров

Областью определения параметров в Eclipse/ExDT является ***контекст (scope)*** оперативной настройки, который определяет:

- список определяемых параметров;
- состав и вид диалога ввода параметров;
- набор и форматы управляющих строк, порождаемых для утилиты.

В Eclipse/ExDT есть иерархия из 4 видов контекстов:

1) ***инсталляционный*** – задает базу «системных» настроек всех утилит Eclipse/ExDT;
2) ***пакетный*** – соответствует «конфигурационным» настройкам утилиты (пакета);
3) ***проектный*** – сооветствует «проектным» настройкам утилиты;
4) ***сеансовый*** – соответствует разовому вызову инструмента.

Контексты появились потому, что существуют программы, параметры которых различаются по степени общности:

- Бывают параметры, отражающие свойства конфигурации всего инструментального пакета в установке на данной машине (например, разрядность или endianness процессора); обычно их значения одинаковы для всех пользовательских проектов на этой машине, но могут меняться при переносе на другую машину. Эти параметры отнесены к *пакетным* настройкам. Некоторые программы принимают такие настройки в виде отдельного *конфигурационного файла*.
- Бывают параметры, которые определяют свойства данного пользовательского проекта (например, архитектура целевой платформы) и должны сохраняться при переносе проекта на другую машину. Эти параметры отнесены к *проектным* настройкам. Некоторые утилиты принимают такие настройки в виде отдельного *проектного файла*.
- Бывают параметры, которые определяют режим конкретного вызова утилиты для выполнения заданной функции (например, отладочный или штатный режим сборки). Эти параметры отнесены к *сеансовым*.

Следует особо подчеркнуть, что все эти параметры управляют только теми утилитами, к командному языку которых относятся; формируемые из них «проектные» и «конфигурационные» файлы являются лишь особыми способами передачи параметров данной утилите, но ни в коем случае не свойствами рабочего проекта и текущей конфигурации Eclipse/ExDT.

- К *инсталляционным* настройкам отнесены параметры, смысл которых выходит за рамки свойств конкретных пакетов, проектов и инструментов: например, свойства инсталляции Eclipse/ExDT на данной машине в целом (имя ОС, принятое в ней расширение имен исполняемых файлов и т.д.). Вряд ли можно ожидать, что эти параметры известны всем утилитам, поэтому их предполагаемое использование – не для непосредственной передачи утилитам (в виде опций), а в условных выражениях языка TSL для задания логики выбора компонент, используемых при данной конфигурации параметров.

Определены следующие ***правила видимости (visibility)*** параметров между контекстами:

- правило ***просачивания (penetration)*** параметров: в приведенной выше иерархии параметры просачиваются сверху вниз, т.е. параметр, определенный в некотором контексте, считается известным без описания также в нижних контекстах иерархии.
- правило ***переопределения (overriding)*** атрибутов: если параметр просачивается в контекст, который уже содержит определение того же параметра, то просачиваются только те его атрибуты, которые не заданы в нижнем контексте.

Например, в проектном файле можно переопределить размер рабочего стека компилятора, заданного по умолчанию в конфигурационном файле, а при конкретном вызове переопределить его еще раз в командной строке.

Основные понятия и объекты TSL

Здесь приводятся определения основных понятий, конструктивных элементов и рабочих механизмов языка TSL, непосредственно используемых для описания динамических настроек инструментов в Eclipse/ExDT.

Модель данных TSL

Элементами TSL-спецификации являются ***объекты***. Объекты имеют:

- ***атрибуты***, выражающие различные свойства объектов;
- ***структуру***, определяющую иерархию описаний объектов (отношение структурной зависимости);
- ***связи***, определяющие различные отношения между объектами (кроме структурной зависимости).

Каталог существующих объектов и их структурная иерархия:

- *интерфейсы*
  - *типы*
  - *форматы опций*
- *контексты*
  - *параметры*
  - *секции ввода*
    - *группы ввода*
  - *секции вывода*
    - *управляющие строки*
- *меню*
  - *меню* (как подменю)
  - *вызовы инструментов*

Данными в TSL являются:

- значения параметров, имеющие входные, выходные и внутренние представления;
- управляющие строки и их элементы – опции, имеющие только выходные представления;

Входные представления задают способ изображения и ввода данных в диалоге настройки.

Выходные представления задают вид, в котором данные передаются утилитам.

Внутренние представления задают вид, в котором данные хранятся в метаданных Eclipse.

Атрибутами объектов могут быть:

- собственное имя объекта и имена объектов, с которыми он имеет связь;
- значения параметров в одном из трех представлений;
- пояснительные тексты в диалогах;
- форматы управляющих строк и опций;
- специальные значения;

Представлениями всех объектов и атрибутов являются **строки** (других данных в XML нет).

Связи между объектами задаются по именам объектов.

Виртуальные типы параметров

Как уже было сказано, значения всех параметров представляются строками. ***Механизм виртуальной типизации параметров*** в TSL позволяет работать со значениями параметров, как с типизированными, т.е. ограничивать множество принимаемых

значений, по-разному осуществлять ввод и преобразование значений между представлениями.

С каждым параметром связывается объект-тип, описывающий эти ограничения. Определены следующие основные виртуальные типы:

- ***логические типы***: возможно определение нескольких типов, различающихся выходными представлениями значений `true` и `false` (например, `"Yes"` и `"No"` или `"+"` и `"-"`);
- ***числовые типы***: допускают ограничение диапазона принимаемых значений и различные способы форматирования значений (например, `"12'333.00"`);
- ***строковые типы***: допускают наложение ограничений на длину строки, задание режима чувствительности к регистру, автокоррекции регистра; как разновидности строкового типа определены типы «файл» и «директория», значения которых запрашиваются диалогом обзора (browse) и представляются в соответствии с правилами данной ОС;
- ***перечислимые типы***: позволяют описывать типы с фиксированным набором значений, вводом значения путем выбора из списка альтернатив, заменой выходных значений.

Языки управления инструментами

Язык управления утилиты формально определяется двумя ***слоями (layers)*** описаний:

1. ***интерфейсом (interface)***, определяющим:
   - именованные *типы (types)* используемых параметров, с атрибутами:
     - *множество значений*, которые может принимать параметр;
     - *выходные представления значений* – формат их записи в управляющей строке;
     - *входные представления значений* – способ и формат их ввода в диалоге настройки;
   - именованные *форматы (formats)* для преобразования параметров в опции управляющих строк;
2. набором ***контекстов (scopes)***, описывающим:
   - все используемые параметры данного контекста, с атрибутами:
     - имена параметра:
       - внутренний *идентификатор (id)*, обозначающий объект-параметр;
       - выходной *идентификатор опции (outid)*, передаваемый утилите;
       - *пояснение* к полю ввода *(label)* в диалоге настройки.
     - имя типа параметра;
     - имя формата для преобразования параметра в опцию;
     - *умолчательное значение* параметра;
     - дополнительные атрибуты параметра;
   - *форматы* всех управляющих строк данного контекста;
   - состав диалога настройки данного контекста;
   - дополнительные специальные компоненты контекста.

Такое расслоение обусловлено тем, что основные типы значений и общие правила записи опций командных языков многих существующих программ либо идентичны, либо сильно похожи (ср. например, Unix-стиль задания опций в виде `-option=value` или MSDOS-стиль `/option:value`). Во всяком случае, можно ожидать, что они будут однотипны для всех утилит из одного пакета. Поэтому в *интерфейс* вынесены те общие элементы командных языков, которые могут использоваться несколькими утилитами. Специфические для утилит элементы языка содержатся в *контексте*.

Интерфейсы

В объекте `interface` перечисляются описания типов параметров и форматы преобразования имен и значений параметров в опции управляющих строк.

Для интерфейсов определен ***механизм наследования описаний***. Каждый интерфейс всегда является производным от какого-то другого (и единственного) ***базового*** интерфейса. В производный интерфейс, в дополнение к его собственным описаниям, просачиваются все описания из его базового интерфейса (включая унаследованные последним).

Механизм наследования позволяет экономить описания, вынося общие элементы в базовые интерфейсы и наследуя их в производные интерфейсы.

По умолчанию базовым интерейсом является предопределенный интерфейс `BasicInterface` (поставляется в файле `BasicInterface.xml`), в котором для общего пользования предлагаются несколько базовых типов и наиболее распространенных форматных стилей. Таким образом, все интерфейсы являются косвенными производными от `BasicInterface` и им доступны его определения. `BasicInterface` – это единственный интерфейс, не являющийся производным от другого.

Контексты

В TSL есть 4 разновидности объекта-контекста: `installation`, `package`, `project` и `tool`, различающиеся только назначением и специфическими атрибутами.

Описание контекстов состоит из следующих секций:

- секция описания параметров, определенных в этом контексте;
- секция `input`, содержащая описание диалога настройки контекста;
- секция `output`, содержащая описание выходных управляющих строк контекста.

Для контекста предусмотрены два вида ***активации (activation)***:

- активация диалога настройки, по которой происходит формирование всех входных представлений параметров и групп ввода, перечисленных в секции `input`, показ диалога настройки, запрос новых значений параметров, типовой контроль введенных значений и сохранение их в метаданных Eclipse;
- активация формирователей управляющих строк, перечисленных в секции `output` контекста, и запись их в выходные файлы.

Первый вид активации выполняется при вызове диалогов настроек в меню инструментов, второй – при выходе из диалога настроек по кнопке Ok, а для контекста `tool` также по кнопке запуска инструмента.

Для контекста `tool` определен ***механизм наследования***. Его назначение в том, чтобы упростить описание инструментов, вызывающих одну и ту же утилиту с незначительной разницей в настройках передаваемых параметров. Основная цель – избавиться от дублирования общих частей громоздких описаний. Механизм позволяет указать некоторый контекст `tool` в качестве основы для структуры производного контекста и затем удалить, добавить или изменить произвольные элементы описания.

Меню инструментов

В ***меню инструментов*** перечисляются доступные инструменты, которые организуются в древовидную иерархию, соответствующую технологии разработки.

Объект `menu` состоит из ***элементов*** `menuitem` (которым соответствуют инструменты) и других объектов `menu`, задающих ***подменю***.

Для меню определен ***механизм редактирования***. Возможно указать некоторое меню в качестве базы для построения производного меню или подменю и при этом удалить, добавить или изменить произвольные элементы.

Шаблоны

Механизм развертки шаблонов (patterns expansion) – это основной и наиболее мощный рабочий механизм TSL, позволяющий гибко формировать одни строковые значения с использованием других. Его основное применение – формирование выходных представлений опций и управляющих строк; кроме этого он используется при формировании списочных и условных значений; при желании его можно использовать и для составления условных пояснений ко вводу.

Терминальная строка (terminal string) – это текст, не содержащий шаблонов.

Форматная строка (format string) – это текст, содержащий шаблоны.

Шаблон (pattern) – это особая подстрока форматной строки, вместо которой в содержащий ее текст должен быть ***подставлен (substituted)*** некоторый другой текст – *значение шаблона*.

Развертка (expansion) форматной строки – это процедура преобразования форматной строки к терминальному виду, заключающаяся в последовательной замене шаблонов их значениями. Развертка шаблона выполняется однократно, без развертки шаблонов, которые могут встретиться в его значении. Это будет пересмотрено.

В TSL есть следующие разновидности шаблонов:

- ***шаблоны-параметры*** – это параметры TSL, определенные в любом из контекстов (включая инсталляционный); значениями этих шаблонов являются просто значения параметров;
- ***шаблоны-генераторы*** – это «псевдопараметры», предопределенные в Eclipse/ExDT, значения которых не настраиваются пользователем, а поступают из операционной среды (см. список в п.5 *«Спецификации TSL»*);
- ***шаблон-повторитель***, служащий для формирования списочных значений;
- ***шаблоны-опции*** – это параметры TSL, используемые в формате управляющей строки; их значениями являются развернутые форматные строки опций.

Условные конструкции

В TSL существуют два вида ***условных конструкций***.

Структурные условные конструкции аналогичны директивам условной компиляции в языках программирования. Они сообщают, что заключенные в их рамки определения должны обрабатываться только при выполнении заданного условия.

Условные выражения аналогичны шаблонам в том смысле, что развертываются в некоторую строку-результат. В условном выражении перечисляются несколько возможных значений и указываются взаимоисключающие условия, при которых результатом условного выражения является каждое из них.

Все условные конструкции интерпретируются динамически, т.е. в момент извлечения значения или в момент обработки описания, в котором стоит условная структурная конструкция. Это позволяет динамически менять структуру описаний: например, скрывать в диалоге настройки параметры, которые не имеют смысла при режиме работы инструмента, заданном другим параметром, а также не передавать в управляющую строку соответствующие опции.

Спецификация TSL

Нотация

При описании синтаксиса TSL используется следующая металингвистическая нотация (РБНФ), в которой метасимволы и метаопределения выделяются шрифтом и цветом:

определяемое понятие → определение – правило определения понятия
[фрагмент **]** – необязательный одиночный фрагмент
{ фрагмент **}** – фрагмент,повторяющийся 0 или более раз
фрагмент **|** … **|** фрагмент – выбор из альтернативных фрагментов

Если понятие определяется в другом разделе, после его имени в скобках указан номер этого раздела.

Формируемый TSL-текст должен подчиняться общему синтаксису языка XML. Вот наиболее важные правила:

- имена тегов и атрибутов чувствительны к регистру;
- каждой открывающей скобке тега `<тег атрибуты>` должна соответствовать закрывающая: `</тег>`
- сокращенная запись: `<тег атрибуты></тег>` ⇔ `<тег атрибуты />` ;
- двойные кавычки, апострофы и знак < в строках: `&qt;` , `≈` , `<`
- имена атрибутов в тегах не должны повторяться.

1. Общая структура TSL-спецификации

Полная ***TSL-спецификация*** настройки Eclipse/ExDT образуется путем слияния описаний из отдельных TSL-файлов. Это все файлы, находящиеся в поддиректории `tools` установочной директории плагина и имеющие расширение **.xml**.

Каждый TSL-файл состоит из описаний одного или нескольких TSL-объектов верхнего уровня: `interface`, `menu` и 4-х видов ***контекстов*** – `installation`, `package`, `project-template` и `tool`, – которые содержат иерархически вложенные описания других объектов более низких уровней.

TSL-файл →

```
<?xml version="1.0" encoding="UTF-8"?>
<elphel-project>
  { интерфейс(2) | меню(3) | контекст(0) }
</elphel-project>
```

Способ разбиения объектов по файлам и порядок их перечисления в файле произвольны; в частности, не важен порядок описания объектов, ссылающихся один на другой. Однако описания объектов верхнего уровня в каждом файле должны быть законченными, и в объединенной спецификации не должно быть разных описаний одного и того же объекта (уникальность объекта характеризуется его тегом и значением атрибута `name`).

2. Интерфейсы

Спецификация ***интерфейса*** задает абстрактное описание языка управления утилитой: ***типы данных***, описывающие значения параметров, и ***форматы опций*** в командной строке утилиты.

интерфейс →

```
<interface name="имя-интерфейса"
          [ extends="имя-базового-интерфейса" ] />
  { тип(2.1) | формат(2.2) }
```

```
</interface>
```

Обязательный атрибут `name` задает уникальное имя интерфейса, используемое для ссылки на него из **контекста(4)** или производного интерфейса.

Необязательный атрибут `extends` задает имя базового интерфейса, используемое для ссылки на другой интерфейс, расширяемый (наследуемый) данным. Если это имя не указано, интерфейс наследует `BasicInterface`. Наследование интерфейса равносильно включению содержимого базового интерфейса в описание производного интефейса. Новые внутренние объекты (типы и форматы) не должны быть одноименны унаследованным (механизмы переопределения при наследовании, существующие для других видов объектов, для интерфейсов не предусмотрены!).

2.1. Типы данных

Тип данных описывает множество возможных значений параметров, способ ввода значений и формат преобразования значения из входного во внутреннее и выходное представления.

определение-типа →

```
<typedef name="имя-типа"
        [ list=("true"|"false") ] >
  структура-типа
</typedef>
```

структура-типа →

числовой-тип(2.1.1) |
логический-тип(2.1.2) |
перечислимый-тип(2.1.3) |
строковый-тип(2.1.4)

Обязательный атрибут `name` задает внутреннее имя типа, используемое для ссылки на тип в **параметре(4.2)** и **перечислимом-типе(2.1.3)**. Все используемые типы должны быть описаны в разделе интерфейса как именованные объекты.

При задании атрибута `list="true"` тип определяется как список, элементы которого описываются **структурой-типа**. Параметры списочного типа имеют составное значение. Ввод списочного параметра производится в отдельном диалоге ввода списка. Для генерации выходного представления списка в форматной строке должен присутствовать **шаблон-повторитель(2.2.1)**.

2.1.1. Числовые типы

Числовой тип определяет диапазон целых чисел или вещественных чисел с фиксированной точкой.

числовой тип →

```
<paramtype kind="number"
           lo="целое"
           hi="целое"
           format="форматная-строка" />
```

Все атрибуты обязательны. Значение нижней границы `lo` должно быть меньше верхней границы `hi`.

Внутреннее представление значения числового типа – строка, содержащая десятичную запись числа цифрами. Атрибуты `lo` и `hi`, а также все значения, используемые при описании утилит, задаются в TSL во внутреннем представлении.

Форматная строка используется для преобразования значения этого типа из входного во внутреннее и из внутреннего в выходное представление. С помощью форматной

строки возможна эмуляция вещественных чисел с фиксированной точкой: в форматной строке указывается точка, отделяющая нужное число десятичных позиций, а все значения масштабируются до целых. – Пока не работает. Вид форматной строки будет доопределен позже; сейчас число вводится и выводится как строка своих значащих цифр.

В диалоге настройки числовой параметр вводится редактируемой строкой по указанному формату. При вводе выполняется контроль границ.

2.1.2. Логические типы

Логический тип имеет традиционную семантику. Может быть описано несколько логических типов, различающихся видом выходных представлений значений.

логический-тип →

```
<paramtype kind="bool"
           formatTrue="строка"
           formatFalse="строка" >
```

Все атрибуты обязательны.

В диалоге настройки логический параметр изображается флажком.

Внутренним представлением значений являются строки `"true"` и `"false"`.

Форматные строки задают вид выходных представлений.

2.1.3. Перечислимые типы

Перечислимый тип задает множество фиксированных значений. Это могут быть избранные значения любого другого типа.

перечислимый-тип →

```
<paramtype kind="enum"
           base="имя-базового-типа">
  { значение-перечисления }
 </paramtype>
```

значение-перечисления →

```
<item value="значение-перечисления"
     [ label="пояснение" ] />
```

Обязательный атрибут `base` задает внутреннее имя базового типа значений, используемых в перечислении. Если значения задаются своим текстовым изображением, рекомендуется использовать в качестве базового строковый тип.

Обязательный атрибут `value` задает значение перечисления, записанное во внутреннем представлении по правилам базового типа. Преобразование в выходное представление выполняется также по правилам базового типа. Необязательный атрибут `label` задает строку, изображающую значение при вводе; если оно не указано, то входное представление строится по правилам базового типа – как правило, оно будет совпадать с заданным в `value`.

В диалоге настройки перечислимый параметр вводится через выпадающий список выбора, в котором показаны все элементы перечисления.

2.1.4. Строковые типы

Строковый тип задает значение произвольного текстового вида.

строковый-тип →

```
<paramtype kind="string"
          [ maxlength="целое" ]
          [ textkind= ("text"|"file"|"dir") ]
          [ sensitivity= ("insensitive"|"sensitive"|"uppercase"|"lowercase") ]
```

```
[ filemask="маска" ] />
```

Необязательный атрибут `maxlength` задает максимальную длину строки (реально это только ограничение длины поля ввода). По умолчанию – 256.

Необязательный атрибут `textkind` указывает специальные разновидности значения строкового типа, определяющие способ его ввода. Обычный текст (по умолчанию, `"text"`) вводится в редактируемом текстовом поле ввода. Имя файла (`"file"`) и имя директории (`"dir"`) вводятся через кнопку обзора; их входное представление формируется по правилам записи имен файлов и путей, принятых в данной ОС.

Необязательный атрибут `filemask` имеет смысл только для файлов и задает маску файлов для диалога обзора.

Необязательный атрибут `sensitivity` определяет преобразование регистра, которым набрана строка. При `"sensitive"` и `"insensitive"` введенная строка не преобразуется; эти два значения устанавливают два режима чувствительности к регистру при сравнении введенной строки со значением атрибута `omit` параметра. При `"uppercase"` и `"lowercase"` строка при вводе преобразуется к верхнему или нижнему регистру; то же происходит с умолчательными значениями, заданными для параметров. Сравнение строк после этого происходит в одинаковом регистре. По умолчанию – `"insensitive"`.

2.2. Форматы опций

Формат опции описывает стиль, которым отдельный параметр и его значение передаются в управляющую строку утилиты.

опция →
```
<syntax name="имя-опции"
        format="формат(2.2.1)" />
```

Все атрибуты обязательны.

Внутреннее имя опции используется для ссылки на формат из **параметра(4.2)**.

Форматная строка управляет формированием выходного представления параметра. Для подстановки в опцию имени параметра и его значения следуют использовать шаблоны-генераторы `%%ParamName` и `%%ParamValue`.

2.2.1. Развертка формата

формат →
```
{ постоянный-текст | шаблон }
```

шаблон →
```
шаблон-генератор |
шаблон-повторитель
```

шаблон-генератор →
```
%%имя-генератора
```

шаблон-повторитель →
```
%( текст-повторитель %| текст-разделитель %)
```

При развертке **формата** за основу берется его текст; затем все шаблоны в нем развертываются по следующим правилам:

1) Шаблон-генератор может развертываться в одиночную строку или список строк. Одиночная строка подставляется значением шаблона. Список строк может быть подставлен только в шаблон-повторитель. Список доступных генераторов приведен в 5.

2) Шаблон-повторитель состоит из двух частей: текста-повторителя и текста-разделителя. В составе текста-повторителя обязателен единственный шаблон-генератор,

поставляющий список строк. Каждая строка этого списка подставляется вместо шаблона-генератора в текст-повторитель; получаемые строки сцепляются вместе, разделяемые(!) текстом-разделителем. Полученная строка замещает шаблон-повторитель в формате.

Пример: шаблон `"%(&qt;%%SourceList&qt;%|;%)"` порождает список файлов проекта, взятых в кавычки и разделенных точкой с запятой.

3. Меню инструментов

Спецификация ***меню*** задает вид меню инструментов.

меню →

```
<menu name="имя-меню"
      label="пояснение-меню"
     [tip="контекстная-подсказка"]
     [icon="файл-иконки"]
     [inherits="имя-меню"]
     [visible=("true"|"false")]
     [after="имя-элемента"] >
  { меню | инструмент }
</menu>
```

инструмент →

```
<menuitem name="имя-элемента-меню"
          label="пояснение-инструмента"
         [icon="файл-иконки"]
         [visible=("true"|"false")]
         [after="имя-инструмента"] >
          call="имя-контекста-инструмента" />
```

Вложенные инструменты и меню задают порядок следования элементов и структур подменю.

Обязательный атрибут `name` задает уникальные имя меню или имя элемента меню, используемые для ссылки на меню и его элементы при редактировании.

Обязательный атрибут `label` задает пояснительный текст к узлу меню или инструмента при отображении.

Необязательный атрибут `tip` имеет смысл только для меню верхнего уровня; он задает развернутый текст контекстной подсказки, выдаваемый при наведении курсора на меню.

Необязательный атрибут `icon` задает путь и имя файла иконки, которой отмечается элемент меню; по умолчанию выдаются стандартные иконки для вызова и подменю.

Обязательный для инструмента атрибут `call` задает имя контекста инструмента, описывающего параметры запуска инструмента.

Необязательный атрибут `inherits` показывает, что описываемое (под)меню создается как производное от базового меню, указанного в этом атрибуте. Описание базового меню должно быть описанием верхнего уровня, т.е. не вложенным.

Механизм редактирования, создающий производное меню из базового, работает следующим образом:

- В изначально пустое производное меню копируется вся структура базового меню, и корень этого меню получает имя производного.
- Затем скопированная древовидная структура сравнивается с описанной структурой производного меню; при этом два узла считаются совпадающими, если их имена одинаковы и их родительские узлы совпадают (в этом же смысле).
- В каждом совпавшем узле выполняются следующие правила редактирования:
  - *правило замены* атрибутов в узле:

- значения атрибутов базового узла заменяются на значения одноименных атрибутов производного узла;
- атрибуты, не указанные в производном узле, сохраняют свое базовое значение;
- атрибуты, не указанные в базовом узле, добавляются из производного;

- *правило удаления и восстановления* ранее удаленных узлов: реализуется через механизм видимости (атрибут `visible`) по правилу замены для этого атрибута;
- *правило добавления узлов*: если у производного узла есть потомки (элементы и подменю), не совпадающие с базовыми, то они добавляются в базовый узел в порядке следования в описании; при этом каждый элемент добавляется:
  - после элемента, указанного его атрибутом `after` ;
  - первым элементом, если `after="first"` ;
  - последним элементом, если `after` отсутствует.

В частном случае: если в описании производного подменю нет других атрибутов, кроме `name` и `inrerits`, базовое меню просто «подшивается» как подменю с указанным именем к его родительскому меню.

4. Контексты

Спецификация ***контекста*** конкретизирует описание языка управления утилитой заданием ***параметров***, видов диалога настройки и управляющих строк, соотвтествующих данному уровню управления утилитой.

```
контекст →
   < ( installation | package | project | tool )
         name="имя-контекста"
         interface="имя-интерфейса"
         специальные-атрибуты(4.1)
   >
     секция-параметров(4.1)
     секция-ввода(4.2)
     секция-вывода(4.3)
     специальные-секции(4.1)
   </( installation | package | project | tool )>
```

Теги контекста соответствуют его разновидности.

Внутренние секции контекста могут задаваться в любом порядке.

Обязательный атрибут `name` задает имя контекста, используемое для ссылки на него из **инструмента**, а также с целью каталогизации контекстов, требуемой для составления списков настройки в меню инструментов Eclipse/ExDT.

Обязательный атрибут `interface` задает имя интерфейса, описывающий типы и форматы опций для параметров данного контекста.

У каждого контекста, дополнительно, есть свои **специальные-атрибуты(4.1)** и **специальные-секции(4.1)**

4.1. Специальные атрибуты и секции контекстов

4.1.1. Специальные атрибуты и секции контекста tool

```
специальные-атрибуты →
      exe="имя-утилиты"
    [ package="имя-контекста-package" ]
    [ project="имя-контекста-project" ]
    [ inherits="имя-контекста-tool" ]
```

Обязательный атрибут `exe` задает имя утилиты – имя исполняемого файла программы, с расширением, без пути. Имя может быть задано условным выражением (6); это

позволяет иметь единое описание контекста для разных ОС, где имена утилит могут различаться.

Необязательный атрибут `package` задает имя контекста-пакета, настройками которого (по правилам видимости) пользуется утилита.

Необязательный атрибут `project` задает имя контекста-проекта, настройками которого (по правилам видимости) пользуется утилита.

Утилита может пользоваться также настройками контекста-инсталляции без дополнительных указаний.

Необязательный атрибут `inherits` задает имя контекста-инструмента, свойства которого наследует данный производный контекст. Правила наследования описаны в 4.5. Этот атрибут несовместен с атрибутами `package` и `project`.

специальные-секции →
секция-расширений

секция-расширений →
`<extensions-list>`
{ расширение }
`</extension-list>`

расширение →
`<extension mask="`*`расширение`*`" />`

Список расширений задает список масок расширений файлов, для которых разрешена активация этого контекста.

4.1.2. Специальные атрибуты и секции контекста project

специальные-атрибуты →
`label="`*`пояснение`*`"`
[`package="`*`имя-контекста-package`*`"` **]**

Обязательный атрибут `label` задает текст пояснения, которым этот контекст идентифицируется в выпадающем списке проектов в меню инструментов Eclipse/ExDT.

Необязательный атрибут `package` задает имя контекста-пакета, настройками которого (по правилам видимости) пользуется этот проектный контекст.

специальные-атрибуты →
пусто

4.1.3. Специальные атрибуты и секции контекста package

специальные-атрибуты →
`label="`*`пояснение`*`"`

Обязательный атрибут `label` задает текст пояснения, которым этот контекст идентифицируется в выпадающем списке пакетов в меню инструментов Eclipse/ExDT.

специальные-секции →
пусто

4.1.4. Специальные атрибуты и секции контекста installation

специальные-атрибуты →
`label="`*`пояснение`*`"`

Обязательный атрибут `label` задает текст пояснения, с которым этот контекст показывается в списке инсталляционных настроек в меню инструментов Eclipse/ExDT

специальные-секции →
пусто

4.2. Секция параметров

Спецификация ***параметра*** полностью описывает входное, внутреннее и выходное представление параметра.

секция-параметров →
{ параметр }

параметр →
```
<parameter id="имя-параметра"
          [outid="имя-опции"]
          [label="пояснение-поля-ввода"]
           type="имя-типа"
           format="имя-формата"
           default="значение"
          [readonly=("true"|"false")]
          [visible=("true"|"false")]
          [omit="значение"]
/>
```

Обязательный атрибут `id` задает имя параметра, которое служит для ссылки на него из **секции-ввода(4.3)** и **секции-вывода(4.4)**. Обычно оно же является выходным именем опции.

Необязательный атрибут `outid` задает выходное имя опции, если оно отличается от имени параметра. (Это бывает необходимо в тех случаях, когда опция многофункциональна и целесообразно разделить ее на несколько параметров, различающихся по смыслу, типу и/или атрибутам).

Необязательный атрибут `label` задает текст пояснения к полю ввода параметра в диалоге. Атрибут может быть опущен для невидимых параметров (`visible="false"`).

Обязательный атрибут `type` задает имя типа, которое служит для ссылки на **определение-типа(2.1)** параметра, задающее множество его значений и форматы представлений.

Обязательный атрибут `default` задает умолчательное значение параметра во внутреннем представлении, совместимое с его типом.

Значение атрибута `readonly="true"` запрещает ввод значения в диалоге. Поле ввода при этом видимо и показывает умолчательное или последнее введенное значение, но не доступно для изменения. По умолчанию - `"false"`.

Значение атрибута `visible="false"` запрещает показ и ввод параметра в диалоге. По умолчанию - `"true"`.

> Атрибут невидимости позволяет задавать параметры с константными значениями, ввод которых не требуется, а также вводить в обиход вспомогательные параметры с условными значениями, которые определяются автоматически в зависимости от значений других параметров.

Необязательный атрибут `omit` задает значение, при котором параметр в выходную строку не передается вообще. (Это необходимо в случае, когда неуказание опции утилите имеет смысл, отличный от указания ее с любым значением, в том числе и умолчательным). В этом случае выходное представление параметра – пустая строка.

Фрагменты списка **параметров** могут заключаться в структурные условные конструкции (6). Строки атрибутов `default`, `omit`, `visible` и `readonly` могут быть условными выражениями (6).

4.3. Секция ввода

В ***секции ввода*** указывается, какие параметры должны быть настраиваемы в диалоге ввода. Параметры могут объединяться в смысловые группы.

секция-ввода →
```
<input [ label="заголовок-диалога" ] >
 { группа-ввода }
</input>
```

группа-ввода →
```
<group [ name="имя-группы" ]
       [ label="заголовок-закладки" ] >
       [ visible=("true"|"false") ]
 { элемент-ввода | секция-вставки | секция-удаления }
</group>
```

элемент-ввода →
```
"имя-параметра"
```

секция-вставки →
```
<insert after="имя-параметра" ] >
 { элемент-ввода }
</insert>
```

секция-удаления →
```
<delete>
 { элемент-ввода }
</delete>
```

Необязательный атрибут **секции-ввода** `label` задает заголовок диалога настройки. По умолчанию это "*context* Preferences", где *context* – вид контекста.

Параметры, объединенные в одну группу, в диалоге настройки выдаются на отдельной закладке в порядке их описания. Порядок закладок в диалоге соответствует порядку описания групп в секции ввода.

Атрибут группы `label` задает заголовок закладки; он обязателен для всех контекстов, кроме `tool`.

Атрибут группы `name` определен только для контекста `tool`; он задает внутреннее имя группы, используемое в производном контексте для редактирования списков параметров и групп (см. 4.5). Если группу не предполагается расширять, его можно опустить. Если значения `name` и `label` одинаковы, `label` можно опустить.

Атрибут группы `visible` задает режим видимости закладки и, соответственно, доступности этой группы параметром для настройки. В контексте `tool` этот атрибут используется для описания производного контекста (4.5). В других контекстах видимостью группы можно управлять с помощью условного значения.

Фрагменты списков **групп-ввода** и **элементов-ввода** могут заключаться в структурные условные конструкции (6). Строка атрибута `visible` может быть условным выражением (6).

Семантика **секции-вставки** и **секции-удаления** описана в 4.5.

4.4. Секция вывода

В ***секции вывода*** указывается, какие управляющие строки передаются утилите из данного контекста и какие параметры участвуют в формировании опций для каждой из этих строк.

секция-вывода →
```
<output>
```

```
    { управляющая-строка }
  </output>

управляющая-строка →
  <line [ name="имя-строки" ]
        [ dest="имя-параметра-назначения" ]
        [ sep="строка-разделитель-опций" ] >
    { управляющий-формат | секция-вставки | секция-удаления }
  </line>

секция-вставки →
  <insert after=управляющий-формат ] >
    { управляющий-формат }
  </insert>

секция-удаления →
  <delete>
    { управляющий-формат }
  </delete>
```

Атрибут `name` определен и обязателен только для контекста `tool`. Он задает внутреннее имя строки, используемое производном контексте для редактирования списков опций и строк (см. 4.5).

Необязательный атрибут `dest` задает имя параметра назначения типа строка-файл (`kind="string" textkind="file"`), значением которого является имя командного файла, куда будет записана управляющая строка. Если атрибут не указан (что допустимо только для одной управляющей строки в списке и только для контекста `tool`), то назначением является командная строка программы.

Управляющая строка формируется путем сцепления строк, получающихся из **управляющих-форматов(4.4.1)**. Если задан атрибут `sep`, то между каждыми двумя смежными строками вставляется строка-разделитель.

Фрагменты списка **управляющих-форматов** могут заключаться в структурные условные конструкции (6).

4.4.1. Формирование управляющей строки по формату

```
управляющий-формат →
    " { постоянный-текст | шаблон } "

шаблон →
    шаблон-опция  |
    шаблон-генератор  |
    шаблон-повторитель

шаблон-опция →
    %имя-параметра

шаблон-генератор →
    %%имя-генератора

шаблон-повторитель →
    %( текст-повторитель %| текст-разделитель %)
```

Порядок формирования управляющей строки по формату аналогичен описанному в 2.2.1. Однако шаблон-параметр здесь работает как шаблон-опция, который развертывается не в значение параметра, а во внешнее представление опции, соответствующее этому параметру. Не допускается использование генераторов, относящихся к свойствам параметров (см.6).

4.5. Спецификация производного контекста tool

Механизм редактирования, создающий производный контекст из базового, указанного в атрибуте `inherits`, работает следующим образом:

- Для атрибутов заголовка производного **контекста(4)**:
  - атрибут `name` должен задавать уникальное имя;
  - атрибут `interface` должен ссылаться либо на интерфейс базового контекста, либо на производный от него;
  - атрибут `exe` может отсутствовать (тогда он наследуется), содержать то же самое имя программы или другое. (Последнее может иметь смысл, если производный контекст обращается к другой версии той же утилиты, инсталлированной в другом каталоге).
- В **секции параметров(4.2)** производного контекста можно:
  - описать новый параметр с уникальным идентификатором;
  - изменить любые атрибуты существующего параметра, переописав их;
- В **секции ввода(4.3)** производного контекста можно:
  - переопределить заголовок диалога – атрибут секции `label`;
  - описать новую группу с уникальным именем и произвольным содержимым;
  - изменить видимость существующей группы (исключить из ввода или включить снова) – атрибут группы `visible`.
  - добавить в группу новые **элементы ввода(4.3)**.

  Механизм редактирования для секции ввода работает следующим образом:
  - Сначала в секцию (группу) добавляются все группы (элементы) из базового контекста в порядке их описания; содержимое уже существующей группы пополняется элементами базы тем же образом.
  - Затем в секцию (группу) включаются все новые группы (элементы) в порядке их описания в производном контексте.
  - Затем, если в группе задана **секция вставки**, то содержащиеся в ней элементы добавляются в эту группу непосредственно после элемента, указанного в атрибуте `after`. Если значение атрибута `after` равно `"first"`, элементы добавляются в начало группы.
  - после этого, если в группе задана **секция удаления**, то содержащиеся в ней элементы удаляются из этой группы.
- В **секции вывода(4.4)** производного контекста можно:
  - описать новую управляющую строку с уникальным идентификатором;
  - удалить строку, переописав ее с `dest=""`;
  - добавить в строку новые **управляющие форматы(4.4)**.

  Механизм редактирования для секции вывода работает аналогично секции ввода.
- В **секции расширений(4.1.1)** производного контекста можно:
  - сейчас ничего

5. Шаблоны-генераторы

Шаблоны-генераторы – это «псевдопараметры», предопределенные в TSL и не требующие описаний и ввода. Их значения поступают из Eclipse/ExDT.

При записи шаблоны-генераторы обозначаются двумя процентами (`%%`), чтобы отличить их от параметров.

В версии Eclipse/ExDT 1.0.2 доступны следующие генераторы (цветом выделены генераторы параметров, недоступные в формате управляющей строки):

- Простые:

  `ParamName` – имя формируемого параметра;
  `ParamValue` – выходное представление значения формируемого параметра;
  `ProjectName` – имя проекта;
  `TopModule` – имя главного модуля (для которого вызвана утилита);
  `CurrentFile` – имя текущего файла (для которого вызвана утилита);

`OS` – имя операционной системы (“Windows” , “Linux” и др.);

- Списочные:

`ParamValue` – выходное представление параметра списочного типа;

`SourceList` – список файлов, содержащих замыкание импортов главного модуля;

6. Условные конструкции

Синтаксис и использование структурных условных конструкций и условных выражений описаны в отдельном документе *«Использование условных конструкций»*.